COMBINATORICS AND ITS APPLICATIONS IN DNA ANALYSIS

SEUNG-BYONG LIGHT GO

Combinatorics and its applications in DNA Analysis

by

© Seung-byong Light Go

A thesis submitted to the

School of Graduate Studies

in partial fulfilment of the

requirements for the degree of

Master of Science

Department of Mathematics and Statistics

Memorial University of Newfoundland

August 2009

St. John's Newfoundland

Abstract

There are several aspects of research in DNA analysis. This thesis is an exploration of four different areas of DNA analysis that use Combinatorics and its applications. First, Levenshtein introduced the idea of Levenshtein Distance. For two strings, Levenshtein Distance is the number of operations (insertions, deletions and substitutions) required to transform one string into the other. An application of Levenshtein Distance includes creation of large sets of synthetic tissue identification that provided error detection and correction. The second area of DNA analysis using Combinatorics is the application of Graph Theory. Two methods of sequencing technique, fragmentation (overlap) method and sequencing by hybridization, both of which use Graph Theory, are studied. The third area of DNA analysis that we study is sequence comparison. Dynamic programming is used to effectively pair up two sequences. A heuristic method of searching sequence alignment such as FASTA is discussed. The final area of DNA analysis studied is the efficient selection of unique oligonucleotide (oligo) from a database containing large DNA or protein sequences. With the large size of database, an effective approach to find unique oligos is required. In this thesis, the Brute-Force method and the filtration methods for the selection of unique oligos, and parallelization of these methods to save some time in searching for unique oligos, are studied. The Brute-Force and filtration methods give us accurate results but they may take a long time. We attempt a new approach, which gives us less accurate results over much improved searching time.

Acknowledgements

Thank God, my salvation and inspiration. I would like to express my gratitude to all people who helped me throughout my studies.

I thank my first thesis advisor Dr. Nabil Shalaby for all his patient help and encouragement.

I thank Dr. Manrique Meta-Montero, my other thesis advisor, for his critiques and help toward the thesis.

I would like to thank the Department of Mathematics and Statistics for providing computer access on Beowulf clusters as well as providing me the Teaching Assistant position during the course of the study.

I would like to thank my family for encouragments and prayers. I thank Rev. John Duff, Dr. John Molgaard and Dr. Gary Sneddon for their encouragements and for always being supportive of me.

Contents

Abstract ii

Acknowledgements iii

List of Tables viii

List of Figures ix

List of Algorithms xi

0 Introduction 1

1 Molecular Biology 5

1.1 DNA 5

1.2 The Central Dogma 8

1.2.1 Replication 8

1.2.2 Transcription 9

1.2.3 Translation 10

1.3 Restriction Enzyme 11

1.4 Expressed Sequence Tag 12

1.5 Mutation 13

1.6 Oncogenes 14

2 Combinatorics and its Applications-Part I:

Coding Theory and Design Theory **15**

2.1 Definitions 16

2.1.1 Hamming Distance 20

2.1.2 Levenshtein Distance (also known as Edit Distance) 21

2.1.3 Error Detecting Codes 22

2.1.4 Error Correcting Codes 22

2.2 Design Theory 25

2.2.1 Balanced Incomplete Block Design 25

2.2.2 Packing Design 27

2.2.3 Covering Design 28

2.2.4 Directed Packing and Directed Covering Design 29

2.2.5 Latin Squares 30

2.3 Construction of Perfect insertion and deletion Codes 31

2.3.1 Length 3 34

2.3.2 Length 4 and 5 36

2.3.3 Length 6 37

2.3.4 Length 7 39

2.3.5 Application of LD 40

3 Combinatorics and its Applications- Part II:

Graph Theory and DNA sequencing **42**

3.1 Introduction to Graph Theory 43
3.2 Fragmentation 47
3.3 Sequencing by Hybridization 53

4 Sequence Alignment **58**

4.1 Complexity 59
4.1.1 Θ-notation 60
4.1.2 *O*-notation 61
4.2 Longest Common Subsequence Problem 61
4.3 Global Alignment 62
4.4 Dynamic Programming Solution 63
4.5 Semi-global Alignment 66
4.6 Local Alignment 67
4.7 Gap Penalty 67
4.8 Heuristics: FASTA 69
4.8.1 Statistical Significance of Alignment Score 69
4.8.2 Hashing 73
4.8.3 FASTA 74

5 Combinatorial Pattern Matching **78**

5.1 Repeats Searching 79
5.2 Exact Pattern Matching 81
5.3 Approximate Pattern Matching 82
5.3.1 Unique Oligonucleotides 83
5.3.2 Algorithms for the unique oligonucleotides design problem . . 85

5.3.3 Parallelization 89
5.3.4 Test Runs 102
5.3.5 More Improvements 106

6 Conclusions **114**

A Algorithms **127**

B Small Example **130**

List of Tables

4.1 Handling Collisions in Hash Table 74

4.2 Hash Table 76

4.3 Table of calculated Q values 77

5.1 Processing time table for parallelized Brute-Force method 104

5.2 Processing time table for filtration method 1 111

5.3 Processing time table for filtration method 2 112

5.4 Number of unique oligos found using different methods 113

B.1 Table for q-mer occurrences in the database. 132

B.2 Hash Table for the small example. 133

B.3 Hash Table for the small example. 135

List of Figures

1 DNA sequence alignment between human and mouse 3

1.1 Deoxyribose with labeled carbon 6

1.2 Molecular Structure of DNA 7

1.3 Replication process 9

1.4 Splicing: messenger RNA 10

1.5 Genetic code 11

1.6 Types of Mutation 14

2.1 Construction of blocks using a diagram. 33

3.1 An example of a graph and a pseudograph 44

3.2 A graph with a trail, a circuit and a cycle. 45

3.3 A non-Eulerian and an Eulerian graph 45

3.4 Hamiltonian and non-hamiltonian graphs 46

3.5 Two diagraphs 46

3.6 Example of trees 47

3.7 Gel Electophoresis 48

3.8 Eulerian graphs for RNA chains 51

3.9 The Eulerian digraph after the partial digestion 52

3.10 Sequence by Hybridization 54

3.11 A problem with Sequence by Hybridization 55

3.12 Graph Theory in Sequence by Hybridization 55

3.13 The Eulerian trails for SBH Spectrum of length 3 56

4.1 Scoring Matrix in Global alignment 64

4.2 Local alignment matrix 68

4.3 Frequency distribution of Q values 77

5.1 An illustration of parallel computing. 90

5.2 Estimating the run time of Algorithm 1 on different numbers of processors. 107

5.3 Estimating the run time of algorithm 2 with the 1-mutant on different numbers of processors. 108

List of Algorithms

1 *Brute-Force algorithm for Pattern Matching* 81

2 Brute-Force algorithm in parallel 92

3 The parallelizing portion of Brute-Force algorithm: each $lMer[[i]]$ will enter into the function for analysis on different computers. 93

4 Parallelized Filtration algorithm 1 95

5 The parallelizing portion of the algorithm 1: each q-mer will enter into this function for analysis on different computers. 96

6 Parallelized Filtration algorithm 2 97

7 The parallelizing portion of algorithm 2: each $qp[[i]]$ will enter into this function for analysis on different computers. 98

8 Needleman-Wunsch algorithm 128

9 Smith-Waterman algorithm 129

Chapter 0

Introduction

Recent developments in technology and informatics have given a significant boost to research in biological sciences. Biologists analyze the interactions of species and the function of cells, researchers depend greatly on collection and analysis of large data such as DNA[1] information to understand the interactions between species [32]. Technological improvements have been allowing us to collect and interpret data faster than ever. Still, many studies and even more technological improvements are needed to handle large amounts of data, so that we can determine which parts of DNA are responsible for the various chemical processes of life. In an attempt to analyze the vast amount of biological data, a new and growing discipline combines other branches of science. Bioinformatics, which is the combination of mathematics, computer science, biology, biochemistry and statistics, is a growing field that attempts to solve biological problems using computers and combinatorics, some of which are explained here, in an attempt to analyze the vast amount of biological data.

[1]DNA is the fundamental substance of which genes are composed.

The term 'gene' defines an inheritable trait that exists in cells. Genes are made of DNA. A gene may be turned on (expressed) or off (not expressed) within a cell. The process of how a gene is turned on or off is called gene expression. Scientists are working very hard to sequence and assemble the genomes of various organisms including humans in an attempt to understand where and how a gene is expressed under normal circumstances.

DNA controls the activity of a gene. Maintenance of DNA is essential for normal function of a cell, and thus to survival [56]. Hormones and other chemical agents in the body of a human try to maintain DNA. However, DNA is constantly damaged by both external and internal agents. It must be repaired by certain mechanisms to maintain the integrity of genetic information [60]. Failure to do so results in mutations which may ultimately result in death. Some of these mutations may be responsible for cancer.

Sequence alignment attempts to identify mutations in genes. Primary sequences of DNA, RNA[2], or protein of an organism can be aligned. Figure 1 is an example of alignment of two DNA sequences. In more detail, the alignment of two sequences is for the purpose of identifying similar regions, which may lead to functional and structural similarities of the two sequences. If two compared sequences are from the same species, mismatches or gaps within the alignment indicate mutations.

There are various types of studies in DNA analysis and most of these studies include different branches of scientific fields. This thesis includes topics from mathematics, computer science and biology.

This thesis is organized as follows: Chapter 1 introduces the reader to the basics of

[2]RNA is a working copy of DNA. Biological terms will be defined in the next chapter.

Figure 1: An example of DNA sequence alignment between human and mouse using EMBOSS Pairwise Alignment Algorithms at http://www.ebi.ac.uk/emboss/align/.

molecular biology. This includes a brief review of DNA, RNA, protein, the processes of replication, transcription and translation. In Chapter 2, we discuss the coding theory and the design theory necessary for the construction of error-detecting and error-correcting codes. We use the notion of Levenshtein Distance (or edit distance), which measures the distance between two strings that are paired (or aligned); this distance is the minimum number of transformations needed to transform one string into the other. We conclude this chapter by giving an applied example of the coding theory and the design theory that uses Levenshtein Distance in gene discovery projects. In Chapter 3, we study Graph Theory applied to DNA sequencing and mapping. We study the Fragmentation method and Sequencing by Hybridization (SBH). In Chapter 4 we study the known methods of aligning DNA and protein sequences. We explore some developed algorithms including FASTA. Finally, in Chapter 5 we review combinatorial pattern matching. We discuss exact pattern matching and approximate pattern matching problems. In particular, our main focus is on the unique

oligonucleotide searching problem that are popularly used in DNA technologies such as microarray [54, 42, 58]. We present the existing brute-force and filtration methods (see [49, 6, 35, 72]) and propose some improvements and a parallelization technique. We implement these algorithms and parallelize them in Mathematica[3]. We give a small example to explain the brute force algorithm, the two filtration algorithms and the parallelized algorithms of each. Running these on the databases of three bacteria species (acaryochloris marina, bacillus cereus and aspergillus nudulans), we find that the parallelization technique works very well and improves the speed of old algorithms.

[3]Mathematica is a fully integrated software environment for technical and scientific computing.

Chapter 1

Molecular Biology

The purpose of this chapter is to provide an overview of gene expression at the molecular level. What is known as the Central Dogma as the organizing theme to define the basics of DNA, RNA and protein is explored here. In particular, this chapter illustrates the application of Bioinformatics in terms of nucleotide sequences.

1.1 DNA

In every organism, the ultimate source of genetic information is stored in nucleic acids. A nucleic acid is a macromolecule made from nucleotide chains. A nucleotide is composed of a nitrogenous base, five-carbon sugar and a phosphate group (Figure 1.1). Nucleotides are joined to each other to form chains called polymers[1] such as DNA or RNA. There are four nitrogenous bases found in DNA: Adenine(A), Cytosine(C), Guanine(G) and Thymine(T). The bases of nucleotides may be classified as either purine or pyrimidine. The bases A and G are purines and C and T are pyrimidines.

[1]A Polymer is a large molecule with repeating chemical structure.

In 1953, Watson and Crick proposed that DNA is formed by two long strands that are entwined giving the shape of a double helix [63]. See Figure 1.2a.

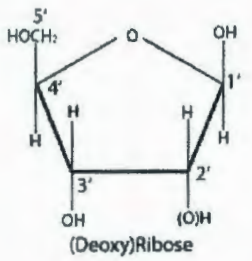
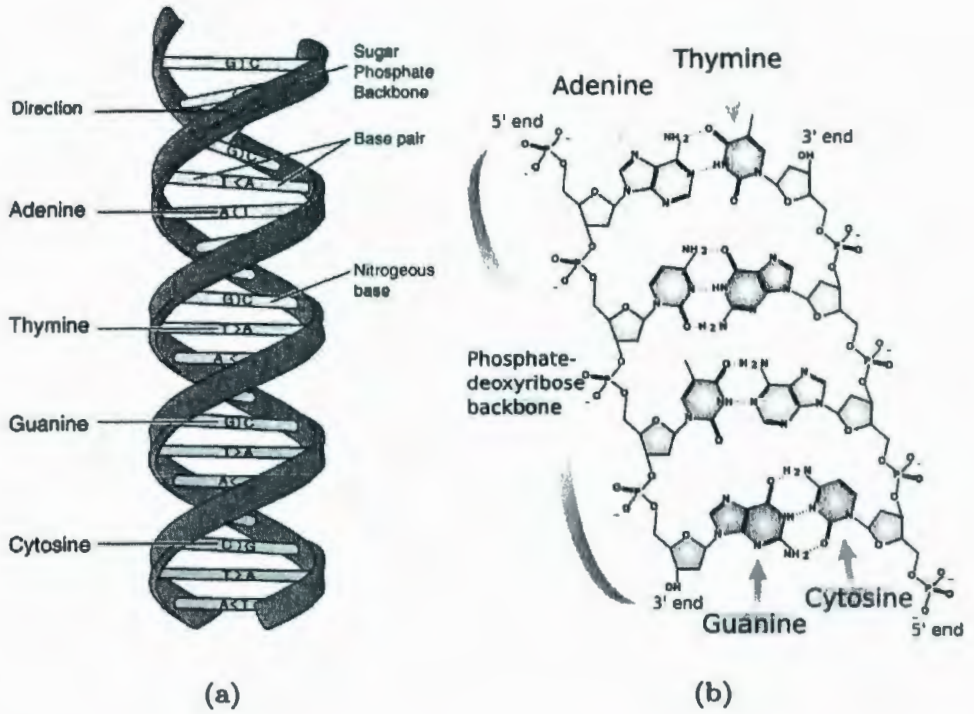


Figure 1.1: Deoxyribose with labeled carbon numerically from $1'$ to $5'$. The labeling is according to the system of naming organic compounds in Organic Chemistry. The above structure composed of five carbons is often referred to as *pentose sugar deoxyribose.*

The 'backbone' of the DNA strand is formed by alternating phosphate and sugar groups. Sugar in DNA is composed of five carbons as in Figure 1.1. Each nucleotide base containing sugars is connected by phosphate groups at the $5'$ and $3'$ ends. The existing bonds by the phosphate groups result in the direction of the nucleotides in one strand. In a double helix, the direction of the nucleotides in one strand is opposite to that of the other. In the sugar backbone of a DNA strand, the $5'$ end is a terminal phosphate group (chemical compound containing phosphate) and the $3'$ end is a terminal hydroxyl group (chemical compound containing OH).

One strand of a DNA chain bonds with the other strand by complementary base pairing. Complementary base pairing refers to nucleotides with the base Adenine that bond with nucleotides with thymine base (A-T). Similarly, nucleotides with the



(a) (b)

Figure 1.2: Deoxyribonucleotides are linked by phosphate group. Sugars, which are read from 5′ end to 3′ end, form the backbone of the DNA. Adenine bonds to Thymine and Cytosine bonds with Guanine in DNA. Figures are adapted from wikipedia [65].

base Guanine bond with nucleotides having the base Cytosine (G-C). As a result of the complementary base pairing, the DNA molecule has entwined helical shape and the two strands of DNA are said to be complementary (see Figure 1.2).

As illustrated in the Figure 1.2, a DNA molecule can be represented by four bases, A, C, G and T. The two strands in DNA are connected by hydrogen bonds between complimentary bases of the two strands. Adenine from the sugar backbone bonds to Thymine and Guanine with Cytosine as in Figure 1.2.

A protein is made up of 20 amino acids, which are polymers. Three nucleotide

bases can be used to identify these amino acids. For example, *CCA* (cytosine-cytosine-adenine) represents Arginine, one of the 20 amino acids. The three nucleotide bases such as *CCA* are called *codons*.

1.2 The Central Dogma

A DNA sequence, which is a nucleotide chain (i.e. a biological polymers) encodes genetic information. The Central Dogma of molecular biology, which was proposed by Francis Crick [19], is a framework for understanding the transfer of genetic information between biopolymers.

There are three types of biopolymers: DNA, RNA and proteins. The transfer of information from DNA to DNA is called *replication*, DNA to RNA is called *transcription* and RNA to protein is called *translation*. The Central Dogma is represented by these three stages; replication, transcription and translation. The dogma, when it was first postulated, highlighted only the three stages, but later it was found that some organisms were able to replicate RNA and even go back to DNA from RNA. For the purpose of this thesis, we will only focus on the three stages of the Central Dogma as it was first proposed.

1.2.1 Replication

To transmit genetic information, the DNA must first replicate [29]. The Central Dogma states that for replication to take place, the two strands of DNA must unwind. The two exposed nucleotide chains act as templates for new strands that are catalyzed

(formed) by the enzyme[2] DNA polymerase (see Figure 1.3). The new chain of DNA is synthesized from the 5′ end to the 3′ end, where free nucleotides are added according to the complementary base pairing of the template strands.

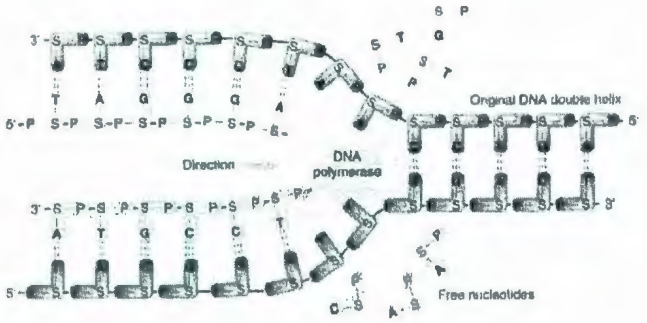


Figure 1.3: Replication Process. This figure is taken from Griffiths [29].

1.2.2 Transcription

An organism is either made of proteins or something that has been made with proteins. Since proteins are encoded in a gene, the products of most genes are specific proteins [29]. The Central Dogma states that to produce a protein from a gene, the cell must copy information encoded in DNA to RNA, which represents a "working copy" of the gene. The process in which nucleotide sequences in DNA are copied onto RNA is called *transcription*. In comparison to DNA, RNA also contains nucleotides containing the sugar ribose instead of deoxyribose in DNA. The RNA contains Uracil (U) instead of Thymine occurring in the DNA. During the transcription process, the DNA acts as a template. A small section of the DNA double helix unwinds and is then used as a template by the enzyme, RNA polymerase to synthesize a messenger

[2]An enzyme catalyzes chemical reactions of the cell.

RNA (mRNA) in a process called splicing (Figure 1.4). The DNA molecule contains parts that code for proteins called exons, and some other parts that do not code anything called introns. In splicing, introns of pre-mRNA are removed and exons of pre-mRNA are joined, resulting in an mRNA molecule. During the transcription process the mRNA is capped (by 7-methylguanine) at the 5′ end and a polyA tail is added at the and 3′ end. The addition of both, a 7-methyguanine cap and a polyA tail, are essential for the proper function of the mRNA.

Since introns do not exist in prokaryotes[3] splicing only occurs in eukaryotes[4]. In the mRNA, genetic information is encoded in the form of triplets of the four possible bases called codons.

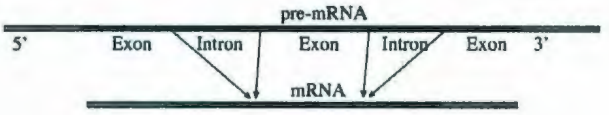


Figure 1.4: Splicing process of the pre-mRNA into the mRNA, where introns are eliminated.

1.2.3 Translation

Translation is the process by which a protein is formed from RNA. The Central Dogma states that a DNA molecule directs its own replication as well as its transcription to form RNA. The sequence of RNA is transcribed into the corresponding amino acids

[3]In general, prokaryotes are organisms that lack a cell nucleus. Bacteria and archaea are prokaryotes.

[4]Eukaryotes are organisms that have cell nucleus. Animals, plants, fungi, and protists are eukaryotes.

which then forms protein.

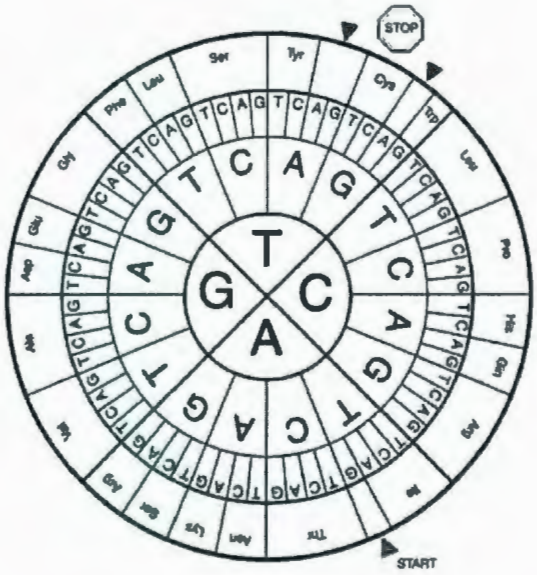


Figure 1.5: Genetic Code: There are three stop codons to mark the end of translation TAA, TAG and TGA for DNA. There is one start codon to initiate translation process: ATG for DNA. This figure is adapted from Bergeron, 2002.

Genetic information is passed in the transcription and translation processes through codons. Since codons are triplets of the four possible bases, then the possible number of codons is $4^3 = 64$. Three codons specify the termination of the chain. Figure 1.5 shows all the possible codons and the respective amino acids.

1.3 Restriction Enzyme

Restriction enzymes cut an RNA sequence after a particular occurrence of a base. There are two kinds of restriction enzymes, the G-enzyme and the U, C-enzyme. The

G-enzyme cuts an RNA sequence after every G base and the U, C-enzyme cuts the sequence after every U or C base. The resulting pieces are called fragments, which we will discuss in a later chapter.

For example, assume we have an RNA chain consisting of $AGGACCGUAAU$. The G-enzyme will break the chain after every appearance of the G base, resulting in the G-fragments: AG, G, $ACCG$, and $UAAU$. Similarly, the U, C-enzyme for the given RNA chain will produce the U, C-fragments: $AGGAC$, C, GU, and AAU.

1.4 Expressed Sequence Tag

In order to study the difference between a normal gene and an altered gene, which may be responsible for a particular disease, researchers must identify and study proteins [1]. The detection of a gene that codes a particular protein is very complicated and time consuming. Sometimes, this process may take years to complete. Often times, attempts to find a gene that codes for a particular protein are only plausible after a certain disease is found. In such cases, scientists can back-track and isolate the location in the chromosome responsible for the construction of the protein causing the disease.

Significant advances in technology such as computers, microarrays (to be discussed in Chapter 5), have helped boost the speed of biological research.

Expressed Sequence Tags (ESTs) are bits of DNA segments which are expressed in a cell. ESTs are short, about 200-800 nucleotides and are generated from either $5'$ or $3'$ end of an expressed gene [47]. An EST serves as a "tag" that can identify unknown genes and to map their position within a genome [1].

The mRNA does not contain non-coding introns; it represents copies from expressed genes. However, mRNA is unstable outside of a cell and it cannot be cloned directly. Instead of using mRNA, scientists use special double-stranded complementary DNA (cDNA) that is generated by reverse transcriptase. These cDNA clones are sequenced to obtain ESTs [47].

1.5 Mutation

Occasionally there may be errors in the DNA replication, transcription and translation processes. For the accurate transmission of genetic information during cell division, these errors must be repaired through a number of DNA repair mechanisms. Failure to do so results in the mutation of a gene.

Sources of errors include ultraviolet(UV) radiation, ionizing radiation, alkylating agents, and/or viruses, which can be present within the cell of an organism, resulting in a base pair sequence change leading to possible mutations.

A mutation is hereditary, which means it can be passed onto offspring. Certain mutations can cause the cell to become malignant, thus leading to cancer [21]. On the other hand, a mutation may lead organisms to better adapt to changes in their environment. For example, a moth may develop offspring with a mutation which changes its color so that it will be harder for predators to detect them.

A chemical mutagen which causes DNA damage can be classified into two major classes: point mutations and insertion/deletion mutations. Point mutations, in which a base pair is replaced by another, can be subclassifed as transitions and transversions. In transitions, a purine is replaced by another purine, $A \leftrightarrow G$ or a pyrimidine is

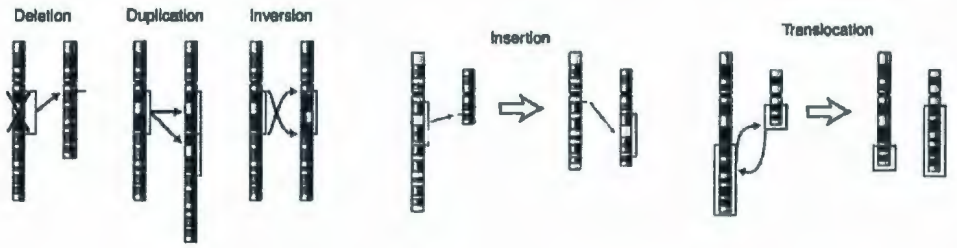


Figure 1.6: Types of Mutation

replaced by another pyrimidine, $C \leftrightarrow T$. In transversions, a purine is replaced by a pyrimidine and vice versa. Insertion/deletion (indel) is where one or more nucleotides are inserted or deleted from the DNA as in Figure 1.6.

1.6 Oncogenes

Oncogenes are mutated forms of genes. They cause normal cells to grow out of control, usually becoming cancerous cells. Oncogenes are mutations of certain normal genes of the cell called proto-oncogenes, which regulate cell division. When there is a mutation of a proto-oncogene, this gene is permanently turned on (expressed). This means that this gene is turned on even when it is supposed to be turned off (inactivated). As a result, there is no control on cell division leading to uncontrolled growth, possibly resulting in cancer [20].

Chapter 2

Combinatorics and its Applications-Part I: Coding Theory and Design Theory

Coding Theory deals with finding an efficient and accurate transfer of information from one place to another. The medium used to transfer information is called a channel. Telephone lines are one example of a channel. Information carried through the channel may be interrupted or disturbed causing the sent information to be different than the received information. These undesirable disturbances, called noise, may be caused by many sources such as weather conditions. Coding Theory analyzes the information that is transferred from the transmitter to the receiver with the noise level as a variable. Implementing the Central Dogma with the concepts of Coding Theory enables scientists to model the transfer of information within organisms using computational techniques [10, 31].

Coding Theory includes five major entities: information source, transmitter (encoder), channel, receiver (decoder) and destination. We rephrase the cellular process in terms of coding theory as follows: The information source contains the genetic information of the human DNA and this is transmitted as a nucleotide sequence through a channel. This channel may be the nucleus of a cell. UV radiation which causes damage in the DNA is a source of disturbance. A strand of mRNA that is expressed from the damaged and un-repaired DNA by the UV light is received from the nucleus to the cytoplasm of the cell. The received message is then sent to the final destination, which is protein synthesis.

The purpose of this chapter is to explore Coding Theory and Design Theory. Later, we introduce Design Theory to discuss directed designs, which may be thought of codewords and error correcting codes of certain lengths. This chapter concludes with an example of an application of Coding and Design Theory to gene discovery.

2.1 Definitions

The information to be sent is often transmitted as a binary sequence of 0's and 1's. A DNA molecule consisting of the four nucleotide bases, A, C, G and T, can be represented using binary sequences, i.e. $A = 00$, $C = 01$, $G = 10$ and $T = 11$. Thus, the sequence $ACGA$ would be represented with 8 bits as 00011000.

Definition 1. Let $\mathcal{A}$ be a finite set of elements. The set $\mathcal{A}$ is referred to as an *alphabet* and its elements are referred to as *letters* or *symbols*. An arbitrary sequence $x = (x_1, x_2, \ldots, x_l)$ of l, l any nonnegative integer, letters of $\mathcal{A}$ is called a word (or a string or an l-tuple) and the number l is called its length. Notice that when $l = 0$,

$x = \epsilon$ is the empty string of length 0.

Example. A DNA sequence (A, T, T, C, G) is a 5-tuple, also a word or a string of length 5, over the alphabet $\{A, C, G, T\}$. The sequence $(0, 1, 1, 0)$ is a 4-tuple over the alphabet $\{0, 1\}$ and this word or string has length 4.

Remark. For simplicity, we may use the notation for strings or sequences without the bracket and commas between the elements. For example, $(0, 1, 1, 0)$ can be simply written as 0110 and (A, G, T) as AGT.

Definition 2. Given two strings x and y we define their concatenation as $x * y$.

Example. Let $x = ACGGT$ and $y = ATTT$ be two DNA sequences. The concatenation $x * y$ is $ACGGTATTT$.

Definition 3. Given two sets of strings A and B we define their concatenation $A*B = \{x * y | x \in A \text{ and } y \in B\}$. In other words, the concatenation of two sets are all the possible concatenations of an element from one set with an element of the other set.

Remark. When $A = B$ we write $A * B$ or A^2 or B^2.

Example. Let $A = \{000, 001\}$ and $B = \{010, 011\}$. The concatenation is $A * B = \{000010, 000011, 001010, 001011\}$.

Definition 4. Let A be a set of strings. The powers of A are: $A^0 = \{\epsilon\}$, $A^1 = A$, $A^n = A^{n-1} * A$, for all $n \geq 1$.

Definition 5. A code C of length n over an alphabet $\mathcal{A}$ of size v is any subset C of $\mathcal{A}_n(v)$ (the set of all n-tuples over $\mathcal{A}$). The number of words of a code is denoted as $|C|$.

Example. A binary code is a set C of words over the alphabet $\{0, 1\}$. The code that has all words of length three is

$$C = \{000, 001, 010, 011, 100, 101, 110, 111\}.$$

The largest possible number of codewords for length three is 2^3.

Remark. A code having all its words of the same length is called a block code. We will consider only block codes.

Definition 6. Let x be a word. If another word y can be obtained from x by deleting some of its letters, y is referred to as subword (or subsequence). Similarly, a word z that is obtained from x by adding some letters to it is referred to as superword (or supersequence).

Example. For $\mathcal{A} = \{0, 1\}$, let $x = 01010$. We can obtain subsequence $y = 001$ from x by deleting second element 1 and the fifth element 0. Inserting 1 and 0 at the start of the sequence x gives the supersequence $z = 1001010$. Also, $w = 01010$ is a superword and a subword of x.

Example. For $\mathcal{A} = \{A, C, G, T\}$, let x be $AACGT$, a DNA sequence. Then, deleting the first and the fifth element of x gives subword $v = ACG$. Deleting the first element of x gives $w = ACGT$. Both v and w are subsequences of x. Adding A before the first element of x gives superword $y = AAACGT$.

Definition 7. Let $x = (x_1, \ldots, x_m)$ be a string over $\mathcal{A} = \{A, C, G, T\}$. A prefix of x is $(x_1, x_2, \ldots, x_i)$, where $0 \leq i \leq m$; when $i = 0$ the prefix is ϵ. A suffix of x is $(x_i, x_{i+1}, \ldots, x_m)$, where $1 \leq i$; when $i > m$ the suffix is ϵ. A substring of x is $(x_i, x_{i+1}, \ldots, x_{j-1}, x_j)$, where $1 \leq i, j \leq m$; when $i > j$ the substring is ϵ.

Example. Let $x = AGGCGTAG$. A prefix of x is $AGGC$. A suffix of x is TAG. $AGGC$ and $CGTA$ are substrings. Also, ϵ is a prefix, suffix and substring of x.

Remark. It is important to note that all substrings are subsequences but not all subsequences are substrings. A substring includes all symbols between two entities of a sequence while a subsequence includes some entities between two symbols of a sequence. The symbols of both subsequences and substrings appear in the same relative orders as in the original sequence. Biologists deal primarily with fragments of DNA which are substrings, not subsequences.

Example. Let $x = ACCAC$. Then ACC is both a subsequence and a substring (also a prefix) of x. However, CCC is only a subsequence, not a substring.

Definition 8. Let $X = \{x_1, x_2, \ldots, x_k\}$ be a collection of strings called input database, where x_i is a sequence over the alphabet $\mathcal{A} = \{A, C, G, T\}$. If $k = 0$, then $X = \{\}$. The length of X is $L = \sum_{i=1}^{k} l_i$, where l_i is the length of x_i.

Example. Let $X = \{x_1, x_2, x_3\}$ such that $x_1 = ACGTA$, $x_2 = ACGTA$ and $x_3 = ACCC$. Clearly, $k = 3$, $l_1 = l_2 = 5$ and $l_3 = 4$. The length of the database, L, is therefore, 14.

If all possible outputs of a channel correspond exactly to the input codes, there is no error present and there is no need to detect errors. In the presence of errors, a channel code must be designed to identify an output code with the correct input code. This is done by recognizing similarities between the output and the input. The idea of similarities between input and output can be formalized using the Hamming Distance(HD) between words.

2.1.1 Hamming Distance

Definition 9. We define addition and multiplication among elements in the set $\mathcal{B} = \{0, 1\}$ to be *mod* 2.

Example.

$$0+0=0,\ 0+1=1,\ 1+0=1,\ 1+1=0,$$

$$0\cdot 0=0,\ 0\cdot 1=0,\ 1\cdot 0=0, 1\cdot 1=1.$$

We generalize addition and multiplication on the set $\mathcal{B}^n$ such that given $x = x_1, \ldots, x_n$ and $y = y_1, \ldots, y_n$, $x + y = x_1 + y_1, \ldots, x_n + y_n$ and $x \cdot y = x_1 \cdot y_1, \ldots, x_n \cdot y_n$

Definition 10. Let g and h be words of length n. The Hamming Distance between g and h, $HD(g, h)$ is the number of positions in which g and h disagree.

Example. If $g = 0110$ and $h = 0111$, $HD(g, h) = 1$. Let $i = ACGT$ and $j = TTTT$ be two DNA codes. The Hamming Distance between i and j is 3.

Similarly, the Hamming Distance can also be determined by obtaining the Hamming Weight between two sequences.

Definition 11. Let g be a binary sequence of length n. The Hamming Weight of g, $HW(g)$, is the number of occurrences of 1 in g.

Lemma. The Hamming Distance between g and h is the same as the Hamming Weight of the error pattern $e = g + h$:

$$HD(g, h) = HW(e).$$

Example. Let $g = 01101111$ and $h = 11001011$.

We have $HD(g, h) = HD(01101111, 11001011) = 3$ and $HW(g+h) = HW(10100100) = 3$.

Definition 12. For a code C containing at least two words the distance of the code is the minimum value of $HW(g + h)$, for all g, h in C.

Example. For $C = \{0001, 0101, 1001\}$, the Hamming Weight of each pair of elements in C is the following.

$HW(0001 + 0101) = HW(0100) = 1$, $HW(0001 + 1001) = HW(1000) = 1$, and $HW(0101 + 1001) = HW(1100) = 2$. Thus, the distance of C is 1.

2.1.2 Levenshtein Distance (also known as Edit Distance)

Levenshtein Distance(LD) or Edit Distance(ED), which was introduced by Levenshtein [43], is the minimum number of substitutions, insertions and deletions (indels) that is required to transform one sequence into another.

Example. Let $x = 011011$ and $y = 110110$. The Hamming Distance between the two strings x and y is four, i.e. $HD(x, y) = 4$. If we take x and delete the first digit 0 and insert 0 after its last symbol we would obtain y, thus, $LD(x, y) = 2$.

Example. Let $z = 012234$ and $h = 5122647$. We cannot obtain the Hamming Distance between the two strings, z and h, because they are of different lengths. If we take z and substitute the first element 0 with 5, substitute the fifth element 3 with 6 and insert 7 at the end of the string, we obtain 5122647, which is h. The transformation from z to h requires these three operations and there is no other shorter way, thus $LD(z, h) = 3$.

2.1.3 Error Detecting Codes

In the next two sections, we follow the notations and definitions from [31].

Definition 13. Suppose that g is in a code C is sent and h in B^n is received, then $e = g + h$ is the error pattern. Any word e in B^n can occur as an error pattern, and we wish to know which error patterns C will detect. We say that code C detects the error pattern e if and only if $g+e$ is not a codeword, for every g in C. In other words, e is detected if for any transmitted codeword g, the decoder, upon receiving $g+e$ can recognize that it is not a codeword and hence that some error has occurred.

Example. Let $C = \{001, 011, 110\}$. Suppose $g = 011$ is sent and the received word is $h = 111$, hence $e = 100$. We calculate $f + e$ for all f in C: $001 + 100 = 101$, $011 + 100 = 111$ and $110 + 100 = 010$. None of the three words 101, 111, or 010 is in C, hence, C detects the error pattern 100. Suppose now $g = 011$ and $h = 001$, hence $e = 010$. We calculate $f + 010$ for all g in C: $001 + 010 = 011$, $011 + 010 = 001$ and $110 + 010 = 100$. Since the first sum 011, and the second sum 001 are in C, we see that C does not detect the error pattern 010. Note that we only require at least one $f + e$ to be in C for the error pattern to remain undetected.

2.1.4 Error Correcting Codes

Definition 14. Let $g \in C$ be transmitted over a channel and h be the received word. Let $e = g + h$ be the error pattern. We say the code C corrects the error pattern e if for all $g \in C$, $g + e$ is closer to g than to any other word in C. By closer, we mean $HD(g, g + e) < HD(d, g + e)$, for all $d \in C$ and $d \neq g$.

Example. Suppose we have code $C = \{001, 101\}$. We wish to see if the code corrects the error pattern $e = 010$.

For $g = 001$, $g + e = 001 + 010 = 011$. $HD(001, g + e) = HD(001, 011) = 1$ and $HD(101, g + e) = HD(101, 011) = 2$.

For $g = 101$, $g + e = 101 + 010 = 111$. $HD(001, g + e) = HD(001, 111) = 2$ and $HD(101, g + e) = HD(101, 111) = 1$.

Since all $g + e$ is closer to g than any other word in C, C corrects the error pattern 010.

Take another error pattern, $e = 110$.

For $g = 001$, $g + e = 001 + 110 = 111$. $HD(001, g + e) = HD(001, 111) = 2$ and $HD(101, g + e) = HD(101, 111) = 1$.

Since $g + e$ is not closer to $g = 001$ than to 101, C does not correct the error pattern 110.

Definition 15. A code C is called a linear code if $f + d$ is a word in C whenever f and d are in C.

Example. A code $C = \{00, 01, 10, 11\}$ is a linear code because all the sums are in C:

$$00 + 00 = 00 \quad 01 + 00 = 01 \quad 10 + 00 = 10 \quad 11 + 00 = 11$$
$$00 + 01 = 01 \quad 01 + 01 = 00 \quad 10 + 01 = 11 \quad 11 + 01 = 10$$
$$00 + 10 = 10 \quad 01 + 10 = 11 \quad 10 + 10 = 00 \quad 11 + 10 = 01$$
$$00 + 11 = 11 \quad 01 + 11 = 10 \quad 10 + 11 = 01 \quad 11 + 11 = 00$$

A code $C = \{00, 01, 10\}$ is not a linear code because $01 + 10 = 11$ is not in C.

The minimum distance of a code measures how "far apart" from each other different codewords are. The minimum distance of a code is defined to be the smallest distance

between two distinct codewords [30].

Lemma. The distance of a linear code is equal to the minimum Hamming Weight of any nonzero codeword.

Definition 16. A code is said to be a t error-correcting code if it corrects all error patterns of Hamming Weight at most t and does not correct at least one error pattern of Hamming Weight $t+1$.

Definition 17. Given non-negative integers n and t, the binomial coefficient is given by,

$$\binom{n}{t} = \frac{n!}{t!(n-t)!}$$

$\binom{n}{t}$ is the number of t-element subsets of an n-element set.

Let g be a word of length n. For $0 \leq t \leq n$, the number of words of length n of distance at most t from g is

$$\binom{n}{0} + \binom{n}{1} + \cdots \binom{n}{t}.$$

Since we are using binary words, there are 2^n words of length n. For $t = n$,

$$\binom{n}{0} + \binom{n}{1} + \cdots \binom{n}{n} = 2^n.$$

Definition 18. A code C of length n and distance $2t+1$ is referred to as a perfect code if,

$$|C| = \frac{2^n}{\binom{n}{0} + \binom{n}{1} + \cdots \binom{n}{t}}.$$

Example. Let $n = 2t+1$.

$$\binom{n}{0} + \binom{n}{1} + \cdots \binom{n}{t} = \frac{1}{2}[\binom{n}{0} + \binom{n}{1} + \cdots \binom{n}{n}] = \frac{1}{2}2^n = 2^{n-1}.$$

$$|C| = \frac{2^n}{2^{n-1}} = 2.$$

Thus, any perfect code of length whose length and distance are both $2t+1$ has exactly two codewords.

2.2 Design Theory

The main references from this section are from Colbourn et al. [18], Street el al. [59] and Wang et al. [61].

2.2.1 Balanced Incomplete Block Design

Definition 19. A balanced incomplete block design (BIBD) is a collection of k-subsets (called blocks) of a v-set V, $k < v$, such that each pair of elements is in V is in exactly λ blocks.

Sometimes, BIBDs are referred to as (v, k, λ) BIBD or (v, b, r, k, λ) BIBD (design), where b is the number of blocks in the design and r is the number of blocks each element in V is included in.

Theorem 1. Given a (v, b, r, k, λ)-design, the following holds.

- 1. $bk = vr$
- 2. $\lambda(v-1) = r(k-1)$

Proof. See Ian Anderson's book [3].

Example. A $(7, 7, 3, 3, 1)$-design is given by the set $V = \{1, 2, 3, 4, 5, 6, 7\}$ and the following 7 blocks ($b = 7$): $\{1, 2, 4\}$, $\{2, 3, 5\}$, $\{3, 4, 6\}$, $\{4, 5, 7\}$, $\{5, 6, 1\}$, $\{6, 7, 2\}$,

and $\{7, 1, 3\}$. We may note that the block size k is 3, and the replication number r is also 3. Each pair of elements appears exactly once in only one block, $\lambda = 1$.

Note that this design is called cyclic since all blocks are obtained by adding 1 (mod 7) to each element of the first block (or developing the block $\{1, 2, 4\}$). The first block is called the base block.

Remark. The condition $v > k$ refers to the "incomplete" in *BIBD* and the parameter of λ refers to "balanced", thus a (v, b, r, k, λ)-design is called *BIBD*.

Definition 20. A directed balanced incomplete block design *(DBIBD)* with parameters (v, b, r, k, λ), is a balanced incomplete block design with parameters (v, b, r, k, 2λ), in which the blocks are regarded as ordered k-tuples and in which each ordered pair of elements occurs in λ blocks. We denote $DBIBD$ by $DB(k, \lambda, v)$.

Remark. Given the block (a, b, c, d) we will say the six ordered pairs (a, b), (a, c), (a, d), (b, c), (b, d) and (c, d) occur in the block.

Example. For $DB(3, 3, 5)$, we have total $b = \frac{5 \cdot 4 \cdot (3 \cdot 2)}{3 \cdot 2} = 20$ blocks. Developing the blocks of $(0, 1, 2)$, $(0, 2, 4)$, $(0, 3, 1)$ and $(0, 4, 3)$ modulo 5, we have the following 20 blocks.

$$\begin{array}{cccc} (0,1,2) & (0,2,4) & (0,3,1) & (0,4,3) \\ (1,2,3) & (1,3,0) & (1,4,2) & (1,0,4) \\ (2,3,4) & (2,4,1) & (2,0,3) & (2,1,0) \\ (3,4,0) & (3,0,2) & (3,1,4) & (3,2,1) \\ (4,0,1) & (4,1,3) & (4,2,0) & (4,3,2) \end{array}$$

Since λ is 3, every pair and the reverse pair are in exactly three blocks, i.e., $(0, 1)$ and $(1, 0)$ are in exactly three blocks.

2.2.2 Packing Design

Definition 21. A packing design $PD(v, k, \lambda)$ is a pair (V, B) where V is a set of v points and B is a family of $k-$subsets from V such that each pair in V is in at most λ blocks. If the number of blocks is maximum, it is called the maximum packing design.

Theorem 2. In a maximum packing design (v, k, λ), the number of blocks, b, is defined by:

$$b \leq \left\lfloor \frac{v}{k} \left\lfloor \frac{(v-1)\lambda}{k-1} \right\rfloor \right\rfloor.$$

Proof. For every point $x \in V$ there are at most $\lambda(v-1)$ pairs including it. If x is in a block, then each block contains $k-1$ pairs including x. Thus, the total number of blocks in which x is included is at most

$$\frac{\lambda(v-1)}{k-1}.$$

This number, however, must be an integer, so it becomes

$$\left\lfloor \frac{\lambda(v-1)}{k-1} \right\rfloor,$$

which includes each v points. These points must all fit in blocks of size k. The total number of blocks becomes,

$$b \leq \frac{v \left\lfloor \frac{(v-1)\lambda}{k-1} \right\rfloor}{k},$$

and it must be an integer thus,

$$b \leq \left\lfloor \frac{v}{k} \left\lfloor \frac{(v-1)\lambda}{k-1} \right\rfloor \right\rfloor.$$

Example. For $v = 6$, $k = 3$, and $\lambda = 1$, we have

$$b \leq \left\lfloor \frac{6}{3} \cdot \left\lfloor \frac{5}{2} \right\rfloor \right\rfloor = \left\lfloor \frac{12}{3} \right\rfloor = 4.$$

The number of blocks in the maximum packing design is four: $\{1,2,3\}$, $\{1,4,5\}$, $\{2,4,6\}$ and $\{3,5,6\}$. Every pair from each block, which is called the leave, is used exactly once except $\{1,6\}$, $\{2,5\}$ and $\{3,4\}$.

2.2.3 Covering Design

Definition 22. A covering design $CD(v,k,\lambda)$ is a pair of (V,B) where V is a set of v points and B is a family of $k-$subsets from V such that each pair in V is in *at least* λ blocks.

Theorem 3. In a minimum covering design (v,k,λ), the number of blocks, b, is defined by:

$$b \geq \left\lceil \frac{v}{k} \left\lceil \frac{(v-1)\lambda}{k-1} \right\rceil \right\rceil.$$

Proof. Similar method as in Theorem 1.

Example. For minimum covering design of $v = 5$, $k = 3$, and $\lambda = 1$, we have $b \geq \lceil \frac{5}{3} \lceil \frac{4}{2} \rceil \rceil = \lceil \frac{10}{3} \rceil = 4$. The four blocks are $\{1,2,3\}$, $\{1,4,5\}$, $\{2,3,4\}$ and $\{2,3,5\}$. Generating pairs from each block, we note that the pair $\{2,3\}$ is used three times but all other pairs are used exactly once. The extra pair $\{2,3\}$ used three times is called the excess graph.

2.2.4 Directed Packing and Directed Covering Design

This subsection is taken from [7]. A directed packing (covering) design with parameters v, k, λ, denoted by (v, k, λ)-DPD ((v, k, λ)-DCD) is a pair (V, B) where V is a set of v points and B is a collection of ordered k-tuples (called blocks) of V, such that every ordered pair of points of V appears in at most (at least) λ blocks of B.

Example. For the DPD we have,

$$b \leq \left\lfloor \frac{v}{k} \left\lfloor \frac{(v-1)2\lambda}{k-1} \right\rfloor \right\rfloor.$$

For $(6, 4, 1)$-DPD, we have $b \leq \left\lfloor \frac{6}{4} \left\lfloor \frac{(5)2}{3} \right\rfloor \right\rfloor = 4$. The four blocks are $(5, 1, 2, 4)$, $(2, 3, 6, 1)$, $(6, 3, 4, 2)$ and $(4, 1, 3, 5)$. We note that there are total $\binom{6}{2} \times 2 = 30$ ordered pairs in this design. From the four blocks, we can only generate $\binom{4}{2} \times 4 = 24$ pairs, thus, there are six directed pairs which are not used which gives us a directed leave graph.

Example. For the DCD we have,

$$b \geq \left\lceil \frac{v}{k} \left\lceil \frac{(v-1)2\lambda}{k-1} \right\rceil \right\rceil.$$

For $(14, 4, 1)$-DCD, let the point set be Z_{13}. We take $(13, 13, 4, 4, 1)$-BIBD on Z_{13} in a decreasing order, that is, the blocks of this design are arranged so that its elements are in a decreasing order. There are 13 blocks in this design. Furthermore we take the following 18 blocks.

$$(0,1,2,12) \quad (0,7,11,13) \quad (13,0,3,9) \quad (0,4,8,10)$$
$$(0,5,6,11) \quad (1,8,9,13) \quad (13,1,4,11) \quad (1,3,7,10)$$
$$(1,5,6,9) \quad (2,3,8,11) \quad (2,4,7,9) \quad (13,2,6,10)$$
$$(2,5,10,13) \quad (3,4,6,13) \quad (3,4,5,12) \quad (13,5,7,8)$$
$$(6,7,8,12) \quad (9,10,11,12)$$

We have total 31 blocks. From the equation we should have $b \geq \left\lceil \frac{14}{4} \left\lceil \frac{(13)2}{3} \right\rceil \right\rceil = \left\lceil \frac{14 \times 9}{4} \right\rceil = 32$ blocks. The above 31 blocks form the desired $(14,4,1)$-DCD on Z_{14} where the ordered pairs $(12,13)$ and $(13,12)$ do not appear any block of the design. There are a total of $\binom{14}{2} \times 2 - 2 = 180$ ordered pairs in this design. From our 31 blocks, we generate $\binom{4}{2} \times 31 = 186$ pairs. We note that there are 6 pairs which are used more than once in the DCD. This gives an excess graph where 6 directed pairs (or edges) are repeated.

2.2.5 Latin Squares

Definition 23. A latin square is an n by n table filled with n sets of the numbers from 1 to n in a way that each number appears exactly once in each row and each column.

Example. A latin square of order 3 is $\begin{vmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \\ 3 & 1 & 2 \end{vmatrix}$.

Definition 24. Two latin squares are orthogonal if the ordered pairs from each position of latin squares are all distinct.

Example. $\mathbf{A} = \begin{vmatrix} 1 & 3 & 2 \\ 2 & 1 & 3 \\ 3 & 2 & 1 \end{vmatrix}$, $\mathbf{B} = \begin{vmatrix} 3 & 1 & 2 \\ 1 & 2 & 3 \\ 2 & 3 & 1 \end{vmatrix}$, $\mathbf{A}$ and $\mathbf{B} = \begin{vmatrix} 13 & 31 & 22 \\ 21 & 12 & 33 \\ 32 & 23 & 11 \end{vmatrix}$.

$\mathbf{A}$ and $\mathbf{B}$ are latin squares and when superimposed, all ordered pairs from corresponding square entries are distinct.

Remark. A set of n latin squares is mutually orthogonal if every pair of latin squares from the set is orthogonal.

2.3 Construction of Perfect insertion and deletion Codes

In this section, we survey constructions of perfect codes. Levenshtein introduced perfect codes of length 3 capable of correcting single deletions [43]. Bours then constructed perfect codes of length 4 and 5, capable of correcting 2 or more deletions [11]. Yin and Shalaby constructed perfect codes of length 6 capable of correcting any combination of up to 4 deletions [69, 55]. Wang et al constructed perfect codes of length 7 capable of correcting 5 deletions [61]. We conclude this section by introducing a paper by Gavin et al which uses synthetic tags to detect and correct errors in codes of length five [26].

Definition 25. Let K be a set of positive integers and let λ be a positive integer. A pairwise balanced design (PBD) of order v and index λ with block sizes from K is a pair (V, B), where V is the point set of cardinality v with a collection B of subsets (called blocks) such that (1)if $D \in B$ then $|D| \in K$ and (2)every pair of V lie in

exactly λ blocks of the PBD.

Example. Blocks for the undirected PBD$(10, \{3, 4\})$ are the following.

$\{1, 2, 3, 4\}$, $\{1, 5, 6, 7\}$, $\{1, 8, 9, 10\}$, $\{2, 5, 8\}$, $\{2, 6, 9\}$, $\{2, 7, 10\}$, $\{3, 5, 10\}$, $\{3, 6, 8\}$, $\{3, 7, 9\}$, $\{4, 5, 9\}$, $\{4, 6, 10\}$, and $\{4, 7, 8\}$.

Definition 26. A group divisible design (GDD) of index λ is a triple (V, G, B), where V is a $v-$set of points, G is a partition of V into subsets (called groups) and B is a collection of subsets of V such that every pair of points from distinct groups appears in exactly λ blocks but no block contains a pair from the same group.

We denote a group-type in exponent terms. $1^a, 2^b, 3^c, \ldots$ means that a group of size 1 occurs "a" times, a group of size 2 occurs "b" times, etc. The notation k-PBD and k-GDD of order v are often used for $\lambda = 1$. A (k, λ)-GDD of type 1^v is a PBD of index λ.

Example. A $(\{3, 4\}, 10)$-GDD of group-type $1^1 3^3$ has groups $\{1, 2, 3\}$, $\{4, 5, 6\}$, $\{7, 8, 9\}$, $\{10\}$ and the following blocks of size three and four: $\{1, 4, 7, 10\}$, $\{2, 5, 8, 10\}$, $\{3, 6, 9, 10\}$, $\{1, 5, 9\}$, $\{2, 6, 7\}$, $\{3, 4, 8\}$, $\{1, 6, 8\}$, $\{2, 4, 9\}$, $\{3, 5, 7\}$. See Figure 2.1.

Definition 27. A directed group divisible design (DGDD) with block size k and order v is a triple (V, G, B) where V is a v-set, G is a partition of V into subsets (called groups), and B is a set of transitively ordered k-subsets (called blocks) of V such that every ordered pair of distinct points of V occurs in exactly one block, and no block meets a group in more than one point. The block size is k and 1^v represents v occurrences of 1 in the multi-set of GDD. Thus, we denote by k-DGDD.

Example. $(5, 2) - DGDD$ of type 3^6 has the groups: $\{0, 6, 12\}$, $\{1, 7, 13\}$, $\{2, 8, 14\}$, $\{3, 9, 15\}$, $\{4, 10, 16\}$, and $\{5, 11, 17\}$. The following are the blocks: $(0, 9, 13, 17, 10)$

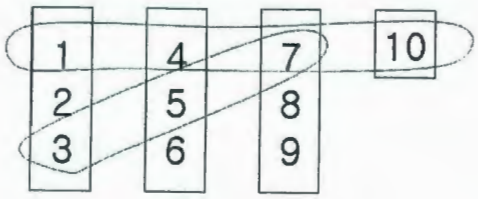


Figure 2.1: A diagram shows how the blocks of size 3 and 4 intersect with the groups. Each block represents a group. Two blocks, $\{1, 4, 7, 10\}$ and $\{3, 5, 7\}$, are constructed as an example. Other blocks can be constructed accordingly. There is no pair formed from the same group.

mod 18, $(0, 14, 16, 3, 1)$ *mod* 18, $(0, 2, 13, 10, 9)$ *mod* 18. This example is taken from [70].

Remark. If a k-GDD exists, then k-DGDD also exists. The k-DGDD is obtained by writing all blocks of the k-GDD twice, once in some order and the other in reverse order. In other words, a (k, λ)-DGDD is a $(k, 2\lambda)$-GDD.

Remark. Recall the following definition. Let v be a positive integer and $\mathcal{A}(v)$ be an alphabet of size v, or equivalently a v-set (of points). By a word of length k over $\mathcal{A}$, we mean a vector (or sequence) of length k with coordinates taken from $\mathcal{A}$. The set of all words of length k over $\mathcal{A}(v)$ will be denoted by $\mathcal{A}_k^*(v)$.

We use $\mathcal{A}_k(v)$ to denote the set of all words of length k over $\mathcal{A}(v)$ with different coordinates (i.e. transitively ordered subsets of size k of $\mathcal{A}(v)$). A subset $C \subseteq \mathcal{A}^k$ is said to be perfect $(k-2)$-deletion-correcting code over $\mathcal{A}$ if every word of $\mathcal{A}^2$ occurs as a subword in exactly one word of C. Such a code is referred to as $T^*(2, k, v)$-code.

Definition 28. A $(k-t)$-deletion/insertion-correcting code over $\mathcal{A}(v)$ is a subset $C^* \subseteq \mathcal{A}_k^*(v)$ (respectively $C \subseteq \mathcal{A}_k(v)$) such that every word in $\mathcal{A}_k^*(v)$ (respectively

$\mathcal{A}_k(v)$) appears as a subsequence of at most one word in C^* (respectively C). The words in C^* (or C) are referred to as codewords.

"$(k-t)$-deletion/insertion correcting" means that we can correct any combination of up to $(k-t)$ deletions and insertions of letters occurred in transmission of codewords. We say the code is capable of correcting $(k-t)$ deletions because any two distinct codewords in C^* (or C) cannot share a common subsequence of length t or longer. This implies that it can correct any combination of up to $(k-t)$ deletions and insertions [61].

There are two types of perfect $(k-t)$ deletion-correcting codes with words of length k over an alphabet of size v; those where the coordinates are equal in size, $T^*(t,k,v)$ and those where the coordinates are different, $T(t,k,v)$. For example, let $\mathcal{A}=\{1,2,3,4,5,6,7\}$ be an alphabet of length 7. Two codewords of equal coordinates of length 3 are $(1,2,3)$ and $(2,5,7)$. Two codewords of different coordinates are $(1,2,2)$ and $(1,2,3)$. Both of T^* and T are capable of correcting any combination of up to $(k-t)$ indels that have occurred during the transmission of codewords. This then can be translated to DNA fragments; let $\mathcal{A}=\{A,C,G,T\}$ be an alphabet of length 4. Two codewords with equal coordinates of length 3 are (A,C,G) and (C,T,A) and two codewords of different coordinates are (A,A,C) and (A,A,T).

2.3.1 Length 3

The notion of perfect deletion-correcting codes was introduced by Levenshtein [43]. In his paper, he proved the existence of a $T(3,4,v)$ code and also proved that the set of permutations of length k can be partitioned into k $T(t-1,t,t)$-codes.

$L(t,n,q)$ is equivalent to $T(t,k,v)$, which is a perfect in $\mathcal{A}_q^n$ code capable of correcting t deletions [43]. Recall that k is the length of the word and v is the size of the alphabet.

Theorem 4. $T(2,3,r)$ exists if $r = 3s$ or $3s+1$, where $s = 1,2,\ldots$

A proof is given in [43]. Let $s = 2p+\alpha+1$, where $\alpha \in \{0,1\}$, and $p = 0,1,\ldots$. Two cases are considered: $r = 3s$ and $r = 3s+1$. Assuming x runs the set B_{r-1}=$\{0,1,\ldots,r-2\}$ (while i and j are fixed) and assuming the addition to be the addition modulo $r-1$. For $r = 3s$ the code is

$$(x, r-1, 3p+1+2\alpha+x),$$

$$(x, 2p+1+2\alpha-i+x, i+x),$$

$$(2j+x, x, 3p+1+2\alpha+j+x),$$

where $i = 1,2,\ldots,p+\alpha$ and $j = 1,2,\ldots,p$. For $r = 3s+1$ the code is

$$(x, r-1, 3p+2-\alpha(2p+1)+x),$$

$$(x, 2p+1+\alpha-i+x, i+x),$$

$$(2j-\alpha+x, x, 3p+2+\alpha+j+x),$$

$$(2p+1+\alpha+y, y, 4p+2+2\alpha+y),$$

$$(2p+\alpha+z, 4p+2+2\alpha+z, z),$$

$$(2p+\alpha+v, v, 4p+2+2\alpha+v),$$

where $i = 1,2,\ldots,p$, $j = 1,2,\ldots,p = 1+\alpha$, $y = 0,1,\ldots,2p+\alpha$, $z = 0,1,\ldots,4p+2\alpha+1$, and $v = 4p+2+2\alpha,\ldots,q-2$.

Example. For $r = 9$, we have $\alpha = 0$, $p = 1$, $i = 1$ and $j = 1$. Using the above construction $T(2, 3, 9)$ would have the following codes:

084 185 286 387 488 580 681 782 883

021 132 243 354 465 576 687 708 810

205 316 427 538 640 751 862 073 184

2.3.2 Length 4 and 5

Bours [11] showed the existence of perfect $(k - t)$ deletion-correcting codes with length 4. He proved the existence of $T(2, 4, v)$ codes for $v \equiv 1(mod\ 3)$ by proving the existence of $DB(4, 1, v)$ for $v \equiv 1(mod\ 3)$.

Definition 29. Let a set E having v elements be given; furthermore let $K = \{k_1, k_2, \ldots, k_n\}$ be a finite set of integers $3 \leq k_i \leq v$, $i = 1, 2, \ldots, n$, and let λ be a positive integer. If it is possible to form a system of blocks (ordered subsets of E) in such a way that:

- 1. the number of elements in each block is some $k_i \in K$,
- 2. every ordered pair of elements of E is contained in exactly λ blocks,

then we shall denote such a system by $B[K, \lambda, v]$.

Let $B = \{b_1, b_2, \ldots, b_k\}$ be an arbitrary block in $B[v, k, \lambda]$. Now take $(b_1, b_2, \ldots, b_k)$ and $(b_k, b_{k-1}, \ldots, b_1)$ as blocks in $DB(k, \lambda, v)$. Take (a, b) and (b, a), where $a, b \in A_n(v)$. Let $B_1, B_2, \ldots, B_\lambda$ be the blocks of $B[v, k, \lambda]$ containing $\{a, b\}$ as a subset, which means that (a, b) will also be a subword of $(b_{i_1}, b_{i_2}, \ldots, b_{i_k})$. Likewise, (b, a) will be a subword of $(b_{i_k}, b_{i_{k-1}}, \ldots, b_{i_1})$. Every word (a, b) is a subword of exactly λ directed blocks of the system. Thus, the system is DB(k, λ, v).

The blocks of these ordered designs can be used for the codewords of the perfect deletion-correcting codes. Every pair of elements of the set F corresponds to a pair of letters of the alphabet, $\mathcal{A}_v$. There is one-to-one correspondence between the elements of the set F and the letters of the alphabet. This unique pair of letters is subword of exactly one codeword of the perfect deletion-correcting code $T(2, k, v)$ (see [11, 43]).

Bours and Mahmoodi [11, 46] presented perfect deletion correcting codes with words of length 4 and 5. For length 4, perfect t-deletion-correcting codes $T(4, 1, v)$ exist for $v \equiv 1(mod\ 3)$ except $v = 1$.

Example. For $v = 19$ the system $DB(4, 1, 19)$ consists of the following 57 words: $(i, i+3, i+12, i+1)$, $(i+13, i+1, i+5, i)$ and $(i+4, i+9, i+6, i)$ modulo 19.

Example. $T(2, 5, 11)$ has the following words: (3,5,1,4,9), (4,6,2,5,10), (5,7,3,6,0), (6,8,4,7,1), (7,9,5,8,2), (8,10,6,9,3), (9,0,7,10,4), (10,1,8,0,5), (0,2,9,1,6), (1,3,10,2,7) and (2,4,0,3,8).

2.3.3 Length 6

Yin [69] presented a combinatorial construction for a $T^*(2, 6, v)$-code capable of correcting any combination of up to 4 deletion and/or insertions of letters that occur in transmission of codewords. In his paper, a $T^*(2, 6, v)$-code exists for some positive integers by using directed group divisible design(DGDD). Later, Shalaby et al [55] proved additional missing cases for $T^*(2, 6, v)$-code by using directed group divisible quasidesign (DGDQD).

Definition 30. A DGDQD is a triple (V, G, B) where V is a finite set (of points), G is a partition of V into subsets (called groups), and B is a collection of sequences

(called blocks) of length k over V with the following properties: (a) every ordered pair of points from distinct groups occurs as a subsequence in exactly one block; (b) for any point x in all but one distinguished group, the pair (x, x) occurs as a subsequence in a unique block, while for any point y in the distinguished group the pair (y, y) does not occur in any block; and (c) all pairs of distinct points from the same group do not occur together in any block.

Example. 6-DGDQD of group type $5^6 2^1$ (i.e. 6 groups of size 5 and 1 of size 2) has the following groups of size 5 plus two infinity points which make up the distinguished group of size 2: $\{j, j+6, j+12, j+18, j+24\}$, for $j = 0, 1, \ldots, 5$. The groups are $\{0, 6, 12, 18, 24\}$, $\{1, 7, 13, 19, 25\}$, $\{2, 8, 14, 20, 26\}$, $\{3, 9, 15, 21, 27\}$, $\{4, 10, 16, 22, 28\}$, $\{5, 11, 17, 23, 29\}$, and $\{\infty_1, \infty_2\}$. There are a total of 80 blocks: (8, 0, 1, 16, 5, 3) *mod* 30, $(1, 0, 14, 10, 21, \infty_1)$ +6 *mod* 30, $(\infty_1, 2, 1, 15, 11, 22)$ +6 *mod* 30, $(3, 2, 16, 12, 23, \infty_2)$ +6 *mod* 30, $(\infty_2, 4, 3, 17, 13, 24)$ +6 *mod* 30, $(5, 4, 4, 18, 14, 25)$ +6 *mod* 30, $(6, 5, 19, 15, 15, 26)$ +6 *mod* 30, $(5, 5, 5, 5, 5, \infty_1)$ +6 *mod* 30, $(\infty_1, 0, 0, 0, 0, 0)$ +6 *mod* 30, $(1, 1, 1, 1, 1, \infty_2)$ +6 *mod* 30, $(\infty_2, 2, 2, 2, 2, 2)$ +6 *mod* 30. The notation +6 *mod* 30 means that 6 *mod* 30 should be successively added to the block, which generates five blocks. Each block intersects with each of the group exactly once. The pair $(8, 0)$ occurs as a subsequence in the first block, however the pair $(0, 8)$ does not occur in any block. This example is taken from [55].

Remark. Adding blocks of the form $(x, x, \ldots, x)$ to a k-DGDD of type $g^t d^1$ yields a k-DGDQD of type $g^t d^1$ where x runs over all groups except for the group of size d. The existence of a k-DGDD implies the existence of a k-DGDQD.

An IDB$(k, 1, g + w, w)$(incomplete DBIBD) can be defined as a k-DGDD of type

$1^v w^1$, in which the group of size w is the hole. The definition of DGDQD can be used to fill in the groups to give $T^*(2,k,v)$-codes. The theorem in [55] states that a $T^*(2,k,v)$ code can be produced with a $T^*(2,6,v)$-code as its subcode by the following, where $v = gt + d + w$. Suppose there exists a k-DGDQD of type $g^t d^1$, $T^*(2,k,v)$-code for a non-negative w and an IDB$(k,1,g+w,w)$. First, adjoin a set F of w infinite points to each group of DGDQD. Replace all groups of size g plus F by IDB to make F as the common hole of size w. Replacing the distinguished group of size $d+w$ by a $T^*(2,k,d+w)$-code, which produces the $T^*(2,k,v)$-code and $T^*(2,6,v)$-code as its subcode.

Example. For $v = 8$, there exists a $T^*(2,6,8)$-code. We take the alphabet to be Z_8. The required codewords are obtained by cycling modulo v the following base codeword, $(0,0,5,0,1,7)$.

Shalaby et al [55] provided the complete proof to show $T^*(2,6,v)$ exists for all positive integers $v \equiv 3(mod\ 5)$ except for $v \in \{173,178,203,208\}$.

2.3.4 Length 7

Wang et al [61] presented a perfect 5-deletion-correcting codes of length 7. $T(2,7,v)$ and $T^*(2,7,v)$ are capable of correcting $(k-t)$ indels. They show this by using DGDD and directed balanced incomplete block design(DBIBD). Recall that a k-GDD of type 1^v is a PBD of index λ. Similarly, a k-DGDD of type 1^v is called DBIBDs denoted by $DB(k,1,v)$. DBIBD is related to $T(2,k,v)$ codes. Given $DB(k,1,v)$, taking a point set X as alphabet and blocks as codewords we have $T(2,k,v)$ code. Conversely, given a $T(2,k,v)$-code, we can form a $DB(k,1,v)$ by reversing the above process. Then,

the necessary and sufficient conditions for the existence of a $T(2,7,v)$ are $v \geq 7$ and $v \equiv 1,7(mod\ 21)$ except for $v = 22$ and possibly for $v = 21t + 1$ and $v = 21t + 7$. In addition, a large number of constructions for $T^*(2,7,v)$-codes with $v < 2350$ were also shown in [61].

2.3.5 Application of LD

Gavin et al[26] presents an application of Levenshtein Distance in a gene discovery program using EST sequencing. In a gene discovery project, identification of tissue source is difficult for the cDNA libraries derived from single tissues. The identification of tissue source becomes even more challenging if the cDNA libraries are derived from multiple tissues. A computer program called UITagCreator by the University of Iowa creates a large set of synthetic tissue identification tags, which provide error detection and correction. The program utilizes a synthetic nucleotide tag to identify the source tissue from which cDNA clone is derived. However, with the presence of errors this program faces more complications. To minimize the complexity of the model program, a process was developed which creates the library tags. From these library tags, identification becomes plausible.

There are three generations of tissue tags. The first generation is composed of the minimum error detection or correction capabilities. Gavin et al[26] obtained kidney tissue to be $CAAAC$ and liver tissue to be $CACAC$. The two sequences differ in their third position giving LD of 1. The second generations of tissue tags have LD of at least 3. The tissue tags for Cerebellum and Hypothalamus have LD of 4. The third generations of tissue tags have the capability of up to two substitution errors.

The use of LD over the Hamming Distance may be better because LD may be more accurate measurement of how close two sequences are. Suppose we have two sequences $GCACT$ and $CACTC$. Hamming Distance measures two sequences of same length. Each position of the sequences are compared and calculated. The Hamming Distance between the two sequences would be 5, whereas the LD is 2.

The creation of tissue tags is important for cDNA clones derived from pooled libraries. Application of the techniques demonstrated by Gavin et al to the biological program may be an important area of research. Gavin et al only considers the ability to detect and correct up to two substitution errors. However, we observe that design theory, along with coding theory, allows codes capable of detection and correction of up to length 7.

Chapter 3

Combinatorics and its Applications- Part II: Graph Theory and DNA sequencing

The entire DNA of a living organism is its genome [49]. Sequencing refers to determining the order in which the four bases occur along a DNA (or RNA) chain. It involves a large number of complex manipulations including chemical processing techniques which for instance break up the DNA into shorter chains, add identifiable groups, and separate and study the fragments as well as computational techniques for handling and analyzing the data. There are numerous methods for sequencing DNA but none of these methods are capable of sequencing the long DNA sequences of most organisms at once. Rather, scientists sequence short DNA segments and map

their positions in the entire genome.

Graph Theory can be used to model the DNA sequencing problem. The study of sequencing techniques and their use of Graph Theory is very important, since Graph Theory may speed-up the process. In this chapter we study two sequencing techniques, the fragmentation (overlap) method and sequencing by hybridization (SBH) [2, 24]. The definitions and theorems from this chapter are adapted from Goodaire and Parmenter, 2006 [28].

3.1 Introduction to Graph Theory

Definition 1. A graph G is an ordered pair (V, E), where V is a set of elements called vertices and E is a family of 2-subsets taken from V called edges. If e is an edge between vertices v and w, we may refer to e as vw or wv. The vertices v and w are said to be incident with the edge vw (or wv).

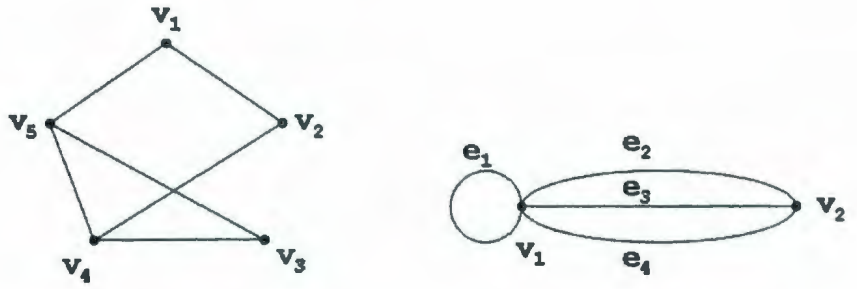
Definition 2. The number of edges incident with a vertex v is called the degree of v. It is denoted as $deg(v)$.

Definition 3. A pseudograph is a graph which may contain multiple edges or loops. A loop is an edge with the same end points i.e. v to v.

Example. Figure 3.1 shows a graph and a pseudograph.

Definition 4. A graph G_1 is a subgraph of another graph G if and only if the vertex and edge sets of G_1 are, respectively, subsets of the vertex and edge sets of G.

Definition 5. A walk is an alternating sequence of vertices and edges, beginning and ending with a vertex, in which each edge is incident with the vertex immediately



(a) An example of a graph (b) A pseudograph

Figure 3.1: An example of a graph and a pseudograph. a)A graph $G(V, E)$ with $V = \{v_1, v_2, v_3, v_4, v_5\}$ and $E = \{v_1v_2, v_1v_5, v_2v_4, v_3v_4, v_3v_5, v_4v_5\}$. b)A pseudograph: The edge e_1 is a loop and e_2,e_3, and e_4 are multiple edges.

preceding it and the vertex immediately following it. A walk, where all edges are distinct is called a trail. A walk where all vertices are distinct, is called a path.

Definition 6. A walk is said to be closed if the start vertex is the same as the end vertex.

Definition 7. A closed trail is called a circuit.

Definition 8. A circuit with no repeating vertices (except the initial and the terminal ones) is called a cycle.

Example. Figure 3.2 shows a graph which contains a trail, a circuit and a cycle.

Definition 9. An Eulerian circuit in a pseudograph (or a graph) is a circuit that contains every vertex and every edge. A pseudograph is Eulerian if it contains an Eulerian circuit.

Example. Figure 3.3 shows two graphs; an Eulerian graph and a non-Eulerian graph.

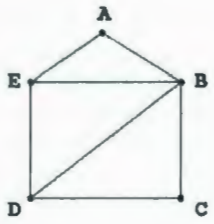
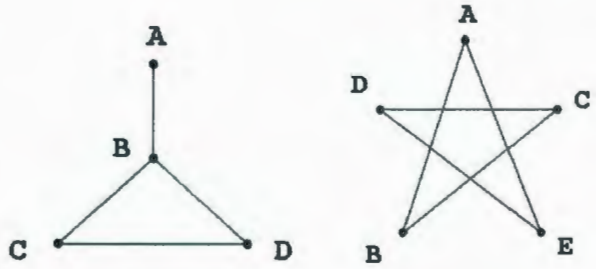


Figure 3.2: $ABCDBE$ is a trail, but not a path. $ABCDEA$ is both a circuit and a cycle.



(a) A non-Eulerian graph **(b)** An Eulerian graph

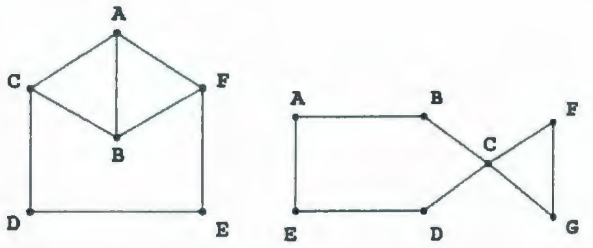
Figure 3.3: a) A circuit starts from vertex A and cannot return to it unless repeating the edge AB. b) A graph that contains an Eulerian circuit $ABCDEA$.

Definition 10. A graph is connected if and only if there exists a walk between any two vertices.

Definition 11. A Hamiltonian cycle in a graph is a cycle including every vertex of the graph. A Hamiltonian graph is one which possesses a Hamiltonian cycle.

Example. Figure 3.4 shows a Hamiltonian graph and a non-Hamiltonian graph.

Definition 12. A digraph consists of two sets V and E, where V is a set of vertices and E is set of ordered pairs called arcs. A digraph is a graph in which each edge has an orientation or direction.



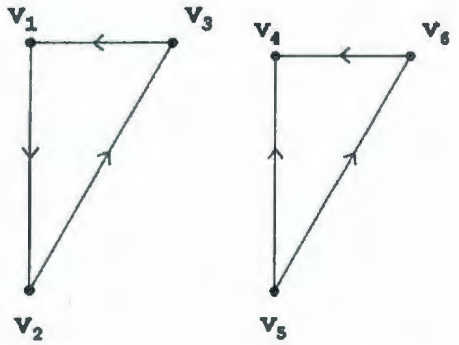
(a) A hamiltonian graph(b) A non-hamiltonian graph

Figure 3.4: a) A Hamiltonian graph. b) A non-Hamiltonian graph.

Example. An example of digraphs is shown in Figure 3.5. Note that with digraphs the term "arc" is used rather than "edge".

Definition 13. A Hamiltonian cycle in a directed graph is a cycle in which every vertex of the graph appears (arcs must be followed in the direction their arrows).

Example. Figure 3.5 shows two different digraphs.



(a) digraph 1 (b) digraph 2

Figure 3.5: a) Two similar graphs with different directed arcs.

Definition 14. The number of arcs (or edges) directed into a vertex is referred to as indegree and that of the arcs directed out of a vertex is referred to as outdegree.

Definition 15. A digraph is called strongly connected if and only if there is a walk from any vertex to any other vertex which respects the orientation of each arc.

Theorem 1. A digraph is Eulerian if and only if it is strongly connected and, for every vertex, the indegree equals the outdegree.

Example. From Figure 3.5, vertex v_4 is indegree 2 and vertex v_5 is outdegree 2.

Definition 16. A tree is a connected graph without any circuits.

Example. Figure 3.6 shows an example of a tree.

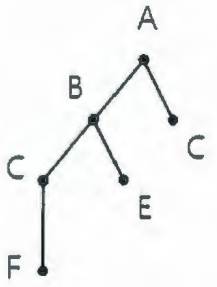


Figure 3.6: Example of a tree.

3.2 Fragmentation

In 1976-77, Allan Maxam and Walter Gilbert developed a DNA sequencing method based on chemical modification of DNA and subsequent cleavage (break) at specific bases. In 1977, Fred Sanger independently developed an alternative method in his published paper "DNA sequencing with chain-terminating inhibitors" [53]. Both

methods require radioactive labeling at one end and purification of the DNA fragment to be sequenced. Chemical treatment generates cleavage at a nucleotide base. A series of labeled fragments are generated, each starting and ending at the cleavage site. Labeled DNA fragments are separated by size by gel electrophoresis. Figure 3.7 is an example of gel electrophoresis. The sequence is read from the bottom, *ATAAAAAACTCAGAACGGCTTCGTA*.

Early sequencing methods dealt with RNA for its simpler single-stranded structure. tRNA is a type of RNA that is involved in translation (Recall from section 1.2.3). tRNA was a good candidate for sequencing because it is short (74-95 bases) and samples are easily obtained. Sequencing tRNA involves the overlapping of smaller fragments of the chain. This overlapping technique was to determine the long RNA chain given just the pieces of short fragments after the chain had been exposed to a complete digest with two enzymes. The technique uses overlaps occurring in the two sets of fragment data to obtain the correct order in which the fragments should be read. This overlap technique is also referred to as the fragmentation method.

Figure 3.7: Gel Electophoresis.

With the collection of fragments of the chain, it is possible to determine the complete enzyme digest. In other words, suppose we do not know the order of a

particular chain but we have the list of the fragments. It is possible to obtain the unknown RNA chain with just fragments using Graph Theory operations. From our previous example in section 1.3 there are 4 possible RNA chains with the G-fragments listed. This possible number of chains must also coincide with the given set of U, C-fragments. The fragmentation method involves using the complete listing of G-fragments and U, C-fragments to construct a digraph and then finding an Eulerian path, which ultimately gives us the unknown RNA chain.

Given the collection of the G- and U, C-fragments lists the procedure to obtain the unknown RNA chain is as follows. From the previous example one might note that the fragment AAU from G-fragments list does not end with G. We refer to such fragment as *abnormal*. An abnormal fragment is the end of the unknown RNA chain. In our example, the RNA chain ends with AAU. If there are multiple candidates for an abnormal fragment, for example a chain that ends with A, then we take the longest abnormal fragment as the end fragment.

Now, we further cut the G-fragments with the U, C-enzyme and U, C-fragments with the G-enzyme. The resulting fragments are called *extended bases*. The extended bases that were neither the first nor the last in their fragments are referred to as *interior extended bases*. Also, fragments that cannot be split by the second enzyme are called *unsplittable fragments*.

Example. From the example in section 1.3 we obtain the following extended bases and unsplittable fragments.

Extended bases from G-fragments: AG, G, AC, C, G, U, AAU

Extended bases from U, C-fragments: AG, G, AC, C, G, U, AAU

Interior extended bases: C, G

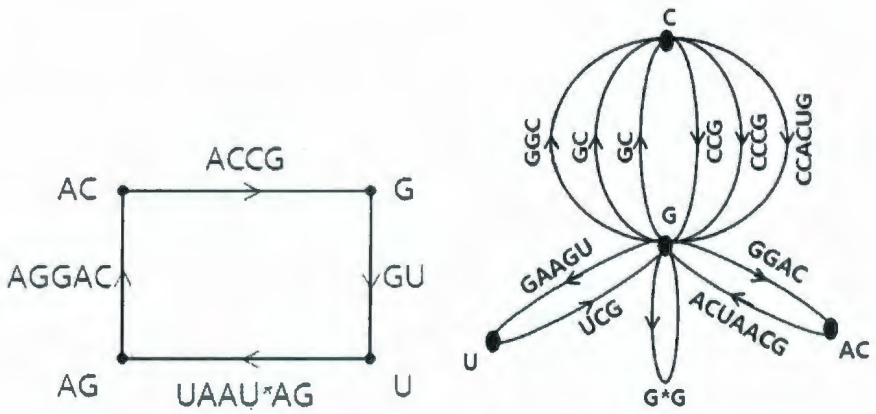
Unsplittable fragments: AG, G, C, AAU

There are exactly two unsplittable fragments that do not belong to interior extended bases. These two fragments are the beginning and the end of the unknown RNA chain. In our example, the two fragments are AG and AAU. We determined AAU to be the end fragment, so AG must be the start fragment.

Now consider all fragments obtained by extended second enzyme. Take a G-fragment $ACCG$, as an example, and draw two vertices and an edge. Here we have a vertex AC and another vertex G. We draw an arc from the first extended base to the last extended base and label the edge with the fragment as shown below.

Repeat constructing arcs for each fragment and matching the overlapping vertices and edges. We draw a final arc from the first extended base of the largest abnormal fragment X to the the first extended base Y of the chain. We label this arc $X * Y$. Then, we determine all Eulerian circuits in the graph which end with $X*Y$. From the previous example, we obtain a directed Eulerian graph as in Figure 3.8a. This gives us the RNA chain, $AGGACCGUAAU$. The original chain is obtained by analyzing Eulerian circuits, knowing the starting and end fragments of the chain. If there are more than one Eulerian circuits, then there is more than one RNA chain. The Eulerian circuits may have a number of possible chains and from the previous example it would be very difficult to determine the particular chain we seek. Figure 3.8b shows

an Eulerian digraph that contains total of 360 possible RNA chains [13]. Partial digest enables the problem to be minimized by exposing the chain to a specific restriction enzyme at different temperatures or limited to a shorter time than for a complete digest. This results in larger fragment size than the complete digest method. Large fragments determine a particular edge with its direction, thus reducing the number of possible chains.



(a) A directed Eulerian graph (b) 360 chains

Figure 3.8: a) Eulerian graph for a RNA chain. Start from the starting fragment AG and travel through directed edge to vertex AC. By doing so, we obtain a chain, $AGGAC$. Repeat until we obtain Eulerian circuit and obtain the complete RNA sequence which is $AGGACCGUAAU$. b) An Eulerian graph that contains 360 possible RNA chains. Adapted from [13].

Example. Suppose we have the following complete enzyme digest with two enzymes: $G-$fragments: G, $ACUG$, ACG, G and $U,C-$fragments: U, $GGAC$, G, GAC which will produce the internal extended bases to be U, and G and unsplittable fragment

to be G, G, U, and G. From this we know that the chain starts and ends with the same G. The Eulerian digraph as in Figure 3.9a. The digraph gives the following 4 possible chains.

$$GGACUGACGG \quad GGACGACUGG$$

$$GACUGGACGG \quad GACGGACUGG$$

Suppose the partial digest with the $G-$enzyme produces the larger fragment of $GACGG$. The correct Eulerian circuit must contain the sequence of arcs GAC, ACG and GG. Knowing these new arcs, we can add new vertex as in Figure 3.9b. The vertex AC' is added to produce GAC and ACG. As a result, we have a reduced graph which results in two possible RNA chains: $GGACUGACGG$ and $GACGGACUGG$ (Figure 3.9b). Additional available data will further reduce the graph resulting in fewer, yet specifically possible RNA chains.

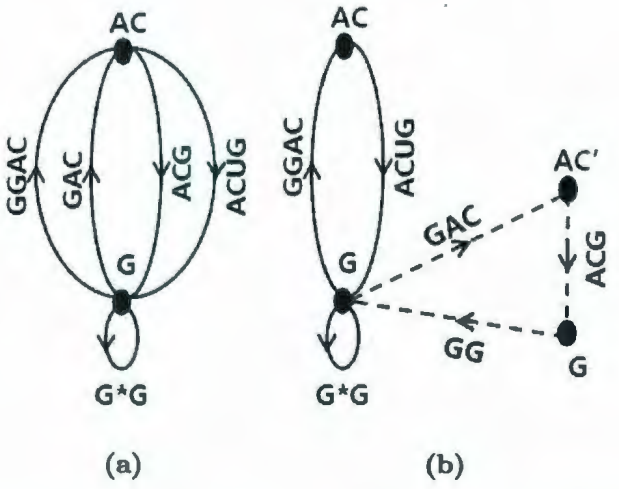


Figure 3.9: a)The Eulerian digraph. b)The reduced Eulerian digraph after the partial digestion.

3.3 Sequencing by Hybridization

Recall that a restriction enzyme from section 1.3 only cut an RNA sequence, not a DNA sequence. The previous fragmentation method for RNA chains is not applicable to DNA sequencing. Sequencing by Hybridization (SBH) was proposed by several research groups in the 80's and is still being developed [8, 45]. SBH is a non-enzymatic method of determining the order in which nucleotides occur on a strand of DNA. In this method, unknown DNA is labeled and compared with the known sequences. If the hybridization (or binding) occurs then that unknown sequence occurs with the known sequence.

In 1988-1989, four groups of biologists independently suggested the SBH. SBH does not involve gel-electrophoresis but rather it involves building a DNA array and combinatorics. An array consists of all possible short sequences of a given length l. A short synthetic fragment of DNA is called a probe. The sequence of nucleotides in a probe is known. These probes are usually about 8 to 30 nucleotides long [49]. They are used to obtain information about an unknown DNA fragment. The DNA strand is composed of four nucleotides, so the array would consist of 4^l subsequences. Given a probe and a single-stranded target DNA fragment, the target will hybridize to the probe if there is a substring of the target that is the complement of the probe. Recall that A is complementary to T and G is complementary to C.

Example. A probe $ATTACGT$ will hybridize with a target $CTAATGCAAT$ since it is complementary to $TAATGCA$ of the target.

The unordered set of all substrings is called the SBH Spectrum of the DNA fragment. For example, the SBH Spectrum (see Figure 3.10) for the target, $ATTAGCC$,

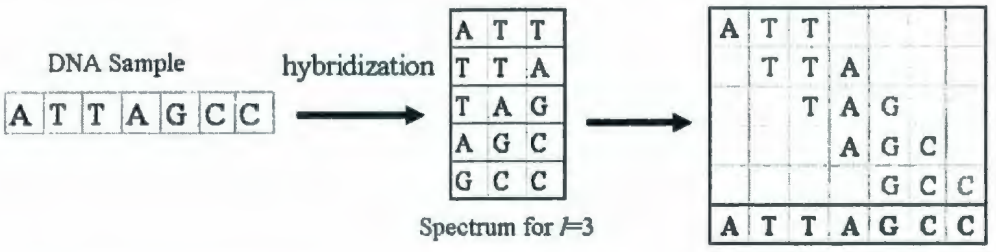


Figure 3.10: A DNA sample is hybridized by spectrum of length 3.

using probes of length three is: $SBH\ Spectrum = \{ATT, TTA, TAG, AGC, GCC\}$. The probes with which the target fragment hybridizes can be detected.

The steps for SBH are as follows. Attach all possible probes of length l to the surface, each probe at a specific location. The array is treated with fluorescence labeled DNA fragment. The DNA fragment hybridizes with those probes that are complementary to substrings of length l of the fragment. Detect probes hybridizing with the DNA fragment with a detector. Then the SBH spectrum of the DNA fragment is obtained[2].

Graph theory and combinatorial algorithms play a key role in understanding and performing SBH reconstruction [2]. In examining the roles that Graph Theory and combinatorics play we assume the ideal case that the number of occurrences of each subsequence of length l in the DNA fragment is known. Determining the DNA fragment using the spectrum requires the listing of each spectrum in order. If a subsequence i is followed by a subsequence j, then the last $l-1$ elements of i match the first $l-1$ elements of j. There may be a problem with SBH for two samples may result in the same spectrum, as in Figure 3.11. Using graphs this could be avoided. There are two ways of representing the list using graphs: Hamiltonian and Eulerian.

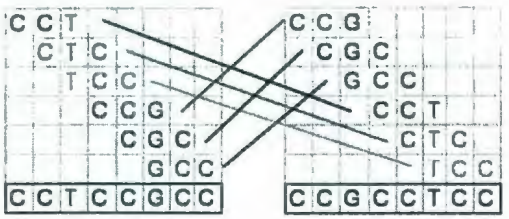


Figure 3.11: There may be a problem with SBH such that two samples may result in the same spectrum as in this figure.

In a Hamiltonian representation, the DNA sequence corresponds to a Hamiltonian path while in Eulerian representation, the DNA sequence corresponds to an Eulerian trail [2]. For the Hamiltonian method, the vertices of the graph are the subsequences in the spectrum. An arc is drawn between two vertices u and v if the last $l-1$ elements in u match the first $l-1$ elements in v. Repeat drawing the arcs until all the spectrum fragments are used. Once all the arcs are constructed, we seek for a Hamiltonian path in the resulting digraph.

Example. Figure 3.12 shows the Hamiltonian path which depicts the following spectrum.

$$Spectrum = \{CAC, ACC, CCT, CTG\}.$$

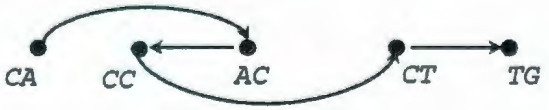


Figure 3.12: Graph Theory in Sequence by Hybridization method. Hamiltonian path starting at vertex CA and ending at vertex TG shows a DNA sequence of $CACCTG$.

Increasing the length of the spectrum increases the complexity of the graphs. There is no efficient algorithm to find Hamiltonian paths in a graph yet [40]. Pevzner [49, 40] reduced the SBH problem to finding Eulerian trails. In this method, the subsequences of length l in the SBH spectrum are arcs in the corresponding graph. The vertices of the graph are the subsequences of length $l-1$ present in the subsequences of length l. For each subsequence in the spectrum, an arc is drawn from a vertex labeled with the first $l-1$ elements of the subsequence to a vertex labeled with the last $l-1$ elements of the subsequence.

Example. Figure 3.13 shows the Eulerian trails for the following SBH Spectrum of $l = 3$.

$$Spectrum = \{ATG, TGC, GCG, GGC, GCA, GGT, TGG, CGT, GTG\}$$

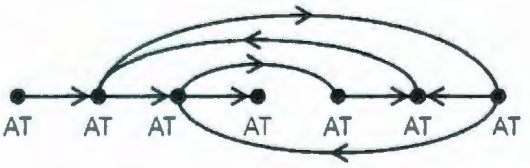


Figure 3.13: The Eulerian trails for SBH Spectrum of length 3.

Note from the previous example that there are two possible Eulerian trails, one giving the sequence $ATGCGTGGCA$ and another one for $ATGGCGTGCA$. With a greater number of spectra, we may have more than one possible DNA sequence. Then it would be difficult to determine the particular DNA sequence we seek. If subsequences of length l are used, it is clear that there will be fewer repetitions of subsequences of length $l-1$, and hence there will be fewer possible DNA sequences [2]. Other

than multiple possible DNA sequences, the SBH method holds many other problems [51]. In the case where hybridization is incomplete, oligonucleotides would be missing from the spectrum. On the contrary, non-specific oligonucleotides may appear in the spectrum leading to wrong DNA sequence. Such errors prevent the SBH method from being an efficient sequencing technique as yet.

Chapter 4

Sequence Alignment

During DNA replication, errors may occur causing insertions, deletions (indels) and substitutions of DNA base pairs, leading to modifications in the DNA nucleotide chains. Such modifications are referred to as mutations. Within an organism, gene comparisons can be done by aligning two sequences. Mismatches between the two sequences might indicate that mutations have occurred. Likewise, matches between two genes from different organisms might indicate the functional or structural similarity between the organisms. Furthermore, it could even allow the prediction of a common evolutionary origin. Gene comparisons are also an application of Levenshtein Distance (LD) as we discussed earlier. In this chapter, we review the different methods of sequence alignment. Sequence alignment using LD finds differences between two sequences being aligned. The Longest Common Subsequence (LCS) algorithm finds similarities between two aligned sequences [49].

When scientists attempt to study the similarities or differences between two strings, they "align" the two strings. One way to align is by using an m by n matrix where

m is the length of one string and n is the length of the other. There are three types of alignment methods: a) Global alignment, which compares the entire length over two strings, b) semi-global alignment, which attempts to find the best possible alignment that includes the start and the end of a sequence and c) local alignment, which attempts to find the best possible alignment that includes conserved regions of similarities.

Finally, we will conclude this chapter by exploring the heuristic for fast database searches that use an idea of filtration. This heuristic includess technique such as FASTA.

4.1 Complexity

Analyzing an algorithm's complexity means establishing the computational resources that the algorithm requires. Often times, different algorithms devised to solve the same problem differ in their efficiency. That is, one algorithm may bring forth the same answer as the other but faster. We wish to measure computational time of an algorithm to identify the most efficient one.

When analyzing an algorithm, it might be possible to determine the exact running time of the algorithm; however, the extra precision may not be worth the effort because of several reasons, including the different speeds of different computers and the time spent to obtain the exact running time. Instead, we can identify an upper bound to the worst-case running time in terms of the input size. In this section we adopt the asymptotic analysis of algorithms; that is, we are concerned with how the running time depends on input size. We review the most general asymptotic

notations, Θ-notation and O-notation. The description of complexity is based on [16].

In general, for a given algorithm, we try to find functions whose domains are the set of natural numbers $N = \{0, 1, 2, \ldots\}$ and are asymptotically greater or asymptotically equivalent to the time complexity of the algorithm.

4.1.1 Θ-notation

Definition 1. For a given function $g(n)$, a function $f(n)$ belongs to the set $\Theta(g(n))$ if there exist positive real numbers c_1 and c_2 and an integer n_0 such that f can be "squeezed" between $c_1 g(n)$ and $c_2 g(n)$, for all $n \geq n_0$. In other words, $0 \leq c_1|g(n)| \leq |f(n)| \leq c_2|g(n)|$, for all $n \geq n_0$.

Remark. $\Theta(g(n))$ is a set, so we take $f(n) = \Theta(g(n))$ to mean the same as $f(n) \in \Theta(g(n))$.

Example. Let $f(n) = 2n^2 + 4n$. We claim $f(n) = \Theta(n^2)$ so that $g(n) = n^2$. We find values for c_1, c_2, and n_0 so that the following holds.

$$\begin{aligned} c_1 n^2 \leq 2n^2 \ &+ \ 4n \leq c_2 n^2 \text{ for all } n \geq n_0. \text{ Divide each side by } n^2 \\ c_1 \leq 2 \ &+ \ \frac{4}{n} \leq c_2. \end{aligned}$$

The right hand inequality holds for any value of $n \geq 1$ by choosing $c_2 \geq 6$. Similarly, the left hand inequality holds for any value of $n \geq 1$ by choosing $c_1 \leq 2$. We can verify that $2n^2 + 4n = \Theta(n^2)$ by choosing $c_1 = 2$, $c_2 = 6$ and $n_0 = 1$. There are other choices for c_1, c_2 and n_0. The point is that there exists a choice of constants which

depend on the particular function $f(n)$. Different functions belonging to $\Theta(n^2)$; say $f(n) = \frac{1}{4}n^2 - 40n + 14$ would have a different choice of constants than the one we have chosen for our $f(n) = 2n^2 + 4n$.

Remark. For an algorithm with time complexity function $f(n)$, if we can find a function $g(n)$ such that $f \in \Theta(g)$ we say that g is equivalent to the time complexity of the algorithm.

4.1.2 O-notation

Definition 2. For a given function $g(n)$, a function $f(n)$ belongs to the set $O(g(n))$ if there exist a positive real number c and an integer n_0 such that $|f(n)| \leq c|g(n)|$ for all $n \geq n_0$.

Remark. We write $f(n) = O(g(n))$ to denote that $f(n)$ belongs to the set of $O(g(n))$.

O-notation describes asymptotic upper bounds. We use it to establish upper bounds to the running time of an algorithm for every input. Thus, O-notation is used to express upper bounds on running time of algorithms. The Θ-notation can be used to establish upper and lower bounds on the running time of an algorithm simultaneously [16].

4.2 Longest Common Subsequence Problem

Needleman and Wunch in 1970 introduced an algorithm that finds the LD of two aligned sequences. Longest Common Subsequence (LCS) Problem is equivalent to

the sequence alignment using Levenshtein insertion and deletion operations (with no substitution) [49].

Let $z(x, y)$ be the length of a LCS of two strings x of length m and y of length n. Then the minimum number of modifications or LD needed to transform x into y (or y into x) is represented by $LD(x, y) = m + n - 2z(x, y)$.

Example. Let $x = ACCGACC$ and $y = AGACC$ be strings over the alphabet $\{A, C, G, T\}$. The longest common subsequence of both x and y is $AGACC$, which has length 5. Thus, $LD(x, y) = 7 + 5 - 2(5) = 2$. We can insert two C's into the second position in y to obtain x or we can delete element C in the second and third positions of x to obtain y.

4.3 Global Alignment

Consider two strings x and y to be aligned, of length m and n respectively. We wish to find an alignment between the two strings such that we maximize the number of matches. We insert gaps denoted by "-" allowing for the possibility of indels.

Remark. We use a superscript to identify the location of the element in a string. Suppose $x = ACGTT$, by x^i we mean substring of x at position i. So, x^2 would be C. Suppose the length of x^2 is 3, then $x^2 = CGT$.

Using the gaps we can then align the strings to form two new strings x_p from x and y_p from y that are equal in length, where $max(m, n) \leq p \leq m + n$. Note that x^i and y^i may not be both gaps.

Example. $AGGCGAT$ and $GCAAGATG$ are aligned below with gaps inserted.

Gaps can occur before the first character of a string, inside a string and after the last character of a string as shown below. If we omit gaps, we would have fewer matchings between two sequences.

$$\begin{array}{cccccccccc} A & G & G & C & - & - & G & A & T & - \\ & & | & | & - & - & | & | & | & \\ - & - & G & C & A & A & G & A & T & G \end{array}$$

In global alignment, similarities between two strings are detected and scored over entire strings. This score is called the raw score.

Definition 3. Raw score ξ is the addition of all weights given to matches, mismatches and gaps, from the alignment of two sequences.

Raw score can give us an approximate idea of how close the sequences are. If a character is aligned with the same character, we say that is a match, represented by α. If a character is aligned with another character, then there is a mismatch, represented by β. Finally, a gap is represented by γ. We assign α, β, and γ the values 1, -1, and -2, respectively. An optimal alignment would maximize $\alpha + \beta, + \gamma$. From the above example of two aligned sequences $AGGCGAT$ and $GCAAGATG$, the raw score is $-2 + (-2) + 1 + 1 + -2 + -2 + 1 + 1 + 1 + (-2) = -5$.

4.4 Dynamic Programming Solution

The Needleman-Wunsch algorithm uses dynamic programming to find a fast solution for the global alignment of two sequences. There are three main steps in dynamic programming: characterize the structure of an optimal solution (initialization), re-

cursively define the value of an optimal solution, and compute the value of an optimal solution in a bottom-up fashion [16].

Let x and y be two strings of length m and n, respectively. The first step in a dynamic programming solution for the global alignment is to create an $m+1$ by $n+1$ matrix. Since the gap was assigned a gap penalty value of -2, we populate the first row and the first column accordingly.

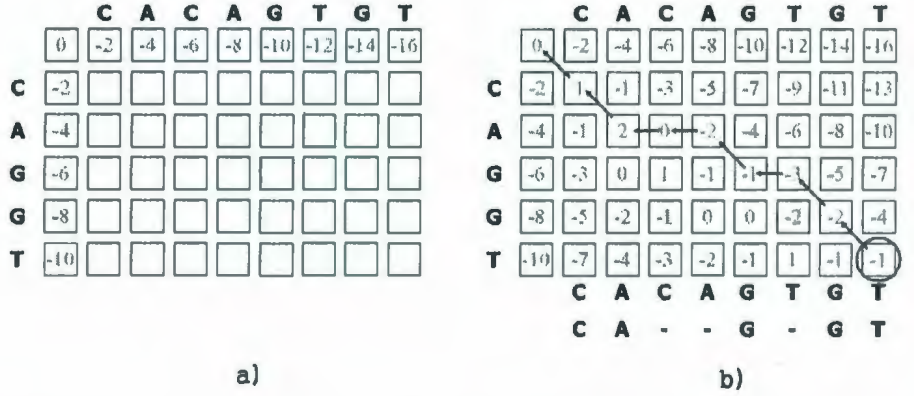


Figure 4.1: a) Two sequences $CACAGTGT$ and $CAGGT$ are paired to form a matrix with initial values. b) Each matrix cell is given its score and the final score is the bottom right corner, -1, where the backtracking begins to obtain the best alignment with gaps inserted in Global alignment. The score for this alignment is $1+1+(-2)+(-2)+1+(-2)+1+1=-1$.

Let $x = CACAGTGT$ and $y = CAGGT$. For each position in the matrix, let $d[i,j]$ be the maximum score at position (i,j), where $p(x_i, y_j)$ is the score of x_i and y_j

occurring as an aligned pair.

$$d[i,j] = max \begin{cases} d[i,j-1] + \gamma, \text{ the gap in sequence x} \\ d[i-1,j-1] + p(x_i, y_j) \\ d[i-1,j] + \gamma, \text{ the gap in sequence y.} \end{cases}$$

Using the above given conditions the score at the $(1,1)$ position can be calculated to be 1, since $d[1,1] = max\{-2,1,-2\} = 1$. The gap penalty is -2, and we fill the rest of the rows and the columns with according values. Using the same techniques, we fill in all values in the matrix as in Figure 4.1b. Once the score values are filled in we can search the matrix to determine the actual alignment that results in the maximum score. The search step begins at the (m,n) position of the matrix. At this (m,n) position, we have a match (ie. T), thus we move on to the next predecessor at the $(m-1,n-1)$ position. We have another match (ie. G) at the $(m-1,n-1)$ position and the next predecessor becomes the entry at the $(m-2,n-2)$ position. At this $(m-2,n-2)$ position, there is a mismatch (T and G), thus, we move to either upward, $(m-2,n-3)$ to -6 or leftward, $(m-3,n-2)$ to -1. We move to the higher value of -1 (over -6). Moving to the leftward or upward indicates that there is a gap in the y or x string, respectively. Repeat this search procedure until we reach the $(1,1)$ position of the matrix. Then the following alignment would be obtained.

$$\begin{array}{cccccccc} C & A & C & A & G & T & G & T \\ | & | & & & | & & | & | \\ C & A & - & - & G & - & G & T \end{array}$$

Note that scoring is typically assigned so that α is positive and β and γ are negative. If we assign α to be 1 and β and γ to be 0, we would obtain the final score at the

(m, n) position to be 5, which is the total number of matches in the above string alignment of x and y.

4.5 Semi-global Alignment

Semi-global alignment is similar to the global alignment. In semi-global alignment we ignore the score of starting gaps, which does not penalize the missing end of the sequence. This scheme works when one sequence is much longer than the other. Finding a gene in a genome is an example of semi-global alignment. In a global alignment, we start with an alignment score of 0. This corresponds to initializing $p(x_i, y_j) = 0$. We can start with a score of 0 after ignoring either the prefix of x or the prefix of y. Also, we ignore end gaps, either a suffix of x or a suffix of y. The matrix scoring for semi-global alignment is obtained by the following rule.

$$d[i,j] = max \begin{cases} d[i-1,j] - 2 \\ d[i-1,j-1] \pm 1 \text{ ,according to match(+1) or mismatch(-1)} \\ d[i,j-1] - 2 \end{cases}$$

Suppose we have two strings of very different lengths, $CAGCACTTGGATTCTCGG$ and $CAGCGTGG$. Using the semi-global alignment, we obtain the following alignment. This alignment has the score of $6(1) + 1(-1) + 1(-2) = 3$.

C	A	G	C	A	C	T	T	G	G	A	T	T	C	T	C	G	G
			|	|	|		|	|	|							
$-$	$-$	$-$	C	A	C	G	T	G	G	$-$	$-$	$-$	$-$	$-$	$-$	$-$	$-$
0	0	0	1	1	1	-1	1	1	1	-2	0	0	0	0	0	0	0

4.6 Local Alignment

Suppose two different strings x and y contain similar regions in the middle. Local alignment aligns a substrings of x and a substring of y, which gives the best score. The matrix scoring for local alignment is obtaining by the following rule.

$$d[i,j] = max \begin{cases} d[i,j-1] - \gamma \text{ the gap in sequence x} \\ d[i-1,j-1] + p(i,j) \\ d[i-1,j] - \gamma \text{ the gap in sequence y} \\ 0 \end{cases}$$

Figure 4.2 gives matrix scoring of the local alignment method of two strings $AGGTATTA$ and $CTATGC$. From Figure 4.2b, the highest score is obtained to be 3. From this cell, we trace back to the highest value toward left, top, or left-top diagonal. The left-top diagonal has the value of 2. Recall that tracing back diagonally means a match.

4.7 Gap Penalty

Recall in global alignment that inserting gaps contributes -2 to the score value of the alignment. In biological applications, mutations cause change in DNA sequences causing insertions or deletions. A single mutational event can create gaps of varying sizes, thus, we treat gaps as a whole rather than individually to avoid high penalties to these mutations[49, 40].

Example. Suppose we align two sequences $ATTAA$ and ATA. The possible alignments are as follows. All three alignments have three matches and two gaps. Recall

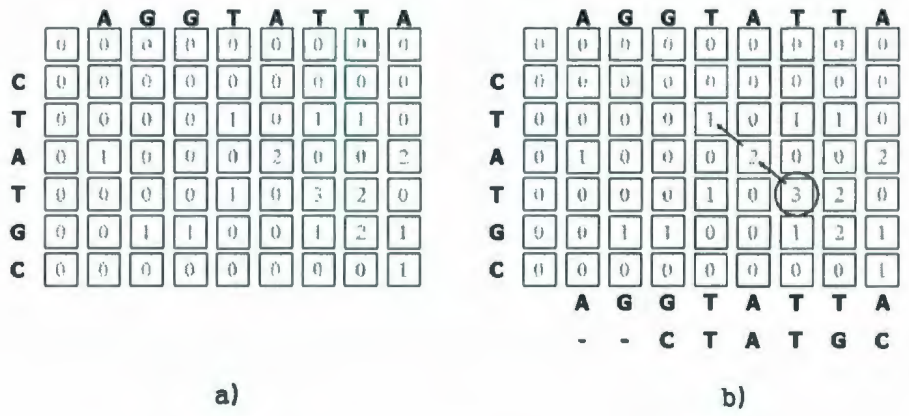


Figure 4.2: a) Alignment scores populate the matrix. b) We trace back to obtain the best alignment in local alignment for the given sequences $AGGTATTA$ and $CTATGC$.

that we use the score of 1 for a match and -2 for a gap.

A	T	T	A	A	A	T	T	A	A	A	T	T	A	A
|	|		|		|	|			|	|	|			|
A	T	$-$	A	$-$	A	T	$-$	$-$	A	A	T	$-$	$-$	A
1	1	-2	1	-2	1	1	-2	-2	1	1	1	-2	-1	1

The scores of each alignment from the left are -1, -1 and 0. The first and the second alignments, which use the regular gap penalties, have the same score. From the biological point of view, gaps occur more contiguously next to each other rather than individually, thus, the second alignment is more plausible over the first one. For this reason, we penalize the gap as a whole as in the third alignment. The opening gap penalty for the third alignment has a score of -2 and the extended gap has the score of -1. The third alignment is preferred over the two previous alignments [49, 40].

4.8 Heuristics: FASTA

A few oncogenes were identified in the early 1980's. Since then, nearly 100 oncogenes which lead to cancer have been identified [71]. With the growing size of the GenBank database, the search for oncogenes and genes involved in normal growth and development has become difficult and time consuming. To search for a string of a gene in a database of size 10^9, we may use a parallel implementation of algorithms.

Many heuristics, such as FAST-A(FASTA) use the idea of filtration for fast database search. The filtration method is used to eliminate or skip certain strings so that the database search may be faster.

An algorithm based on dynamic programming performs similarity searches based on local sequence alignment. Obtaining an optimal local alignment, one with highest score, between some strings turns out to be very difficult. To avoid the high expense involved in finding an optimal alignment, heuristics have been developed. The description in this section is from [68].

4.8.1 Statistical Significance of Alignment Score

Definition 4. A segment pair (x, y) consists of two strings, x and y, of the same length.

The Smith-Waterman algorithm will find all the segment pairs between two strings whose scores cannot be improved by extensions. This is referred to as a high-scoring segment pair (HSP). In order to analyze how a score is measured, a random sequences model is constructed. Recall that a DNA sequence is composed of A, C, G and T. Let P_A denote probability that the nucleotide A occurs in a sequence. Given P_A,

P_C, P_G and P_T in a database of DNA strings, the probability of a source sequence x having a perfect match with the target sequence, y is given by:

$$p(x) = \prod_{i=1}^{n} P_{x_i}.$$

Example. The probability of $x = AATCCG$ having a perfect match to a target sequence y is $p = P_A^2 P_C^2 P_G P_T$.

Let L_y be the target sequence length and L_x be the source sequence length. The number of possible matching operations (the number of possible alignment between two sequences) is given by:

$$n = L_y - L_x + 1.$$

Example. Let $x = AT$ and $y = TCATGG$. We have $L_y = 6$, $L_x = 2$ and $n = 5$. There are five possibilities where AT of x can be matched under y: TC, CA, AT, TG, or GG.

The binomial probability distribution of the number of matches is given by $pd(x)$ where p is the probability of a successful match and $q = 1 - p$ is the probability of failure [9].

$$pd(x) = \underbrace{\frac{n!}{(n-x)!x!}}_{\alpha} \underbrace{p^x q^{n-x}}_{\beta}$$

α is the number of outcomes with exactly x successes among n trials and β is the probability of x successes among n trials for any one particular order.

If the probability of each codon is one in a total of four (for DNA, i.e., $P_A = P_C = P_G = P_T = 0.25$), from the above example we have $p = 0.25^2 = 0.0625$, $q = 0.9375$, $L_y = 6$, $L_x = 2$ and $n = 5$. The probabilities of having $0, 1, \ldots$ exact matches of two

letters (rounded to four decimals) are:

$$\begin{aligned} pd(0) &= \frac{5!}{(5-0)!0!}p^0q^{5-0} = 0.7241 \\ pd(1) &= 0.2414 \\ pd(2) &= 0.0322. \end{aligned}$$

The probability of having at least one match is simply: $1 - pd(0) = 1 - 0.7241 = 0.2859$. The calculation of the binomial probability distribution can be complicated and tedious as n gets very large. The Poisson probability distribution can be used to approximate the binomial probability distribution since the computations involved in calculating binomial probabilities are greatly reduced [38, 9].

Definition 5. The Poisson probability distribution with parameter ($\lambda = np$) is given by:

$$\begin{aligned} \varphi(x) &\approx \frac{n^x}{x!}p^x e^{-np} \\ &= \frac{(np)^x}{x!}e^{-np} \\ &= \frac{\lambda^x}{x!}e^{-\lambda} \text{ , where } \lambda = np. \end{aligned}$$

Remark. Approximating the binomial probability distribution gives the Poisson probability distribution.

$$\begin{aligned} \varphi(x) &= \frac{n!}{(n-x)!x!}p^x q^{n-x} \\ &= \frac{n(n-1)(n-2)\dots(n-x+1)}{x!}(\frac{p}{q})^x(1-p)^n \\ &\approx \frac{n^x}{x!}p^x e^{-np} \end{aligned}$$

Example. From the above example we have $p = 0.25^2 = 0.0625$, $q = 0.9375$, $L_y = 6$,

$L_x = 2$ and $n = 5$. The Poisson probability distribution for $x = 0$ is,

$$\varphi(0) = \frac{\lambda^x}{x!}e^{-\lambda} = \frac{[(5)(0.0625)]^0}{0!}e^{-(0.0625)(5)} = 0.7316.$$

Given x, y, L_y, L_x and let us assume $P_A = P_C = P_G = P_T = 0.25$. The probability of finding an exact match of at least R consecutive letters is $p = 0.25^R$, where $R \leq L_y, L_x$. If the size of a source sequence and a target sequence are m and n, respectively, then there are mn possible matching operations. Thus, the expected matches with length at least R is given by:

$$E = mn0.25^R = mn2^{-2R} = mn2^{-\xi},\ \xi = 2R.$$

According to Atschul-Dembo-Karlin, the number of matches with score above ξ is approximately Poisson distributed,

$$E = Kmne^{-\lambda\xi},$$

where K and λ are scaling constants.

Recall that raw score ξ is the sum of the alignment's pair-wise scores using a specific scoring matrix (see earlier sections in this chapter). The probability of matching exactly x with a score greater than or equal to ξ is given by

$$pm = e^{-E}\frac{E^x}{x!}$$

The probability of matching at least one HSP "by chance" is given by

$$pm = 1 - p_0 = 1 - e^{-E}.$$

This is the p-value associated with the score ξ. If we expect to find two $HSPs$ with score greater than or equal to ξ, the probability of matching at least one is 0.95.

For a given $HSP(x, y)$ the raw score ξ is converted into bit score, Ψ, which has been normalized with respect to the scoring system. Bit score can be compared between different alignments [39, 38, 22].

Definition 6. Bit score is given by the following [39].

$$\Psi = \frac{\lambda\xi - ln(K)}{ln2}$$

The significance of a given bit score can be determined by obtaining its own E-value [39, 38, 22].

$$E = Kmne^{-\lambda\xi} = Kmne^{\Psi ln(2)+ln(k)} = mn2^{-\Psi}.$$

4.8.2 Hashing

This subsection is based on [16].

Hashing is the transformation of a string of characters into a shorter string (often into binary numbers of 0's and 1's) or keys. Given a large database, we create the hashes for the data, store the hashes and these hashes can be compared if we wish to compare the large amounts of data.

If we have a collection of n elements with a set $\{0, 1, \ldots, m-1\}$, where $m \leq n$, then we can store the items in a direct address table, $T[0, 1, \ldots, m-1]$ indexed by keys. Using a direct address table may not be efficient as the collection of n elements becomes very large. An efficient way is to use a hash function, $h(k)$ which maps the set of keys into a certain space formed with slots. This results in a hash table which contains the actual database information. As a hash table is generated there may be more than one data that can be mapped into the same slot in the hash

table. This is referred to as collisions. There are several ways to avoid collisions (see Table 4.1). Hashing DNA sequences can be done in different ways. One way is to use

Handling Collisions			
Technique	**Method**	**pros**	**cons**
Chaining	chain all collisions in lists attached to the appropriate slot in the hash table	handle large number of elements and collisions	overhead of multiple linked lists
Re-hash	re-hash until an empty slot is found	fast	possibly multiple collisions
Overflow	shift the collision into the reserved area	fast	possibly multiple collisions

Table 4.1: Ways to handle collision problem.

a nucleotide as an index and another way is to define an l-mer[1] and convert it into a binary sequence.

4.8.3 FASTA

FASTA (fast-all) is a heuristic for finding significant matches between two strings, x and y. Its general strategy is to find the most significant paths in the dynamic programming matrix or dot-plot. The FASTA algorithm consists of two main steps. First, the algorithm determines all exact matches of length k between the two se-

[1]Recall that an l-mer is a substring of size l.

quences. To find these exact matches, it uses a special form of hashing (look-up). This creates a hash table which consists of all words of length k that are contained in the sequence x. H_y values, which are the locations of each word of length k from the database sequence y, are recorded. Then Q values, which is the difference between the position of each word in x and H_y values, are also recorded. The second step is compiling a frequency distribution of Q values. The highest frequency number tells the number of position the sequence y should shift in order to obtain the maximum number of matches between x and y.

Example. Given query sequence x, and the database sequence y are as follows:

	0	1	2	3	4	5	6	7	8	9	0	1	2
x	A	G	C	T	G	G	A	A	G	G	C	A	T
y	A	G	G	A	A	G	C	C	A	T	C	C	T

We find H_y values and construct a table as in Table 4.2, and we then obtain Q values as in Table 4.3. For example, nucleotide A, which is a word of length 1 occurs at positions 0, 3, 4 and 8 in sequence y. Q values for A at position 0 are $0-0=0$, $0-3=-3$, $0-4=-4$ and $0-8=-8$. Then, we would not look for the Q values for A at position 3, but instead consider the next word in x, which is G. Q values for G at position 1 are $1-1=0$, $1-2=-1$ and $1-5=-4$.

The number of values equal to 0 is 5, which indicates that there are five matched nucleotides between two sequences x and y without shifting any position (we see they match at positions 0, 1, 5, 10, and 12). We can use these values of differences in table to construct a graph (see Figure 4.3). From this graph, we can then observe the highest frequency. The values equal to 3 occur the most (eight times), which

Bases	Position, H_y values	Total
A	0 3 4 8	4
C	6 7 10 11	4
G	1 2 5	3
T	9 12	2

Table 4.2: Hashing of databases sequence y in FASTA algorithm. H_y is the location of corresponding base (A, C, G and T) and the total number of its appearances. For example, A appears at the positions 0, 3, 4 and 8 in sequence T.

indicates that shifting y three positions to the right would give us the most matches of eight between two sequences x and y.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
x	A	G	C	T	G	G	A	A	G	G	C	A	T			
y				A	G	G	A	A	G	C	C	A	T	C	C	T

Source x	A	G	C	T	G	G	A	A	G	G	C	A	T
Position	0	1	2	3	4	5	6	7	8	9	10	11	12
Q values	0	0	-4	-6	3	4	6	7	7	8	4	11	3
	-3	-1	-5	-9	2	3	3	4	6	7	3	8	0
	-4	-4	-8		-1	0	2	3	3	4	0	7	
	-8		-9				-2	-1			-1	3	

Table 4.3: Calculated Q values which is the difference between the location number of the source sequence x and H_y.

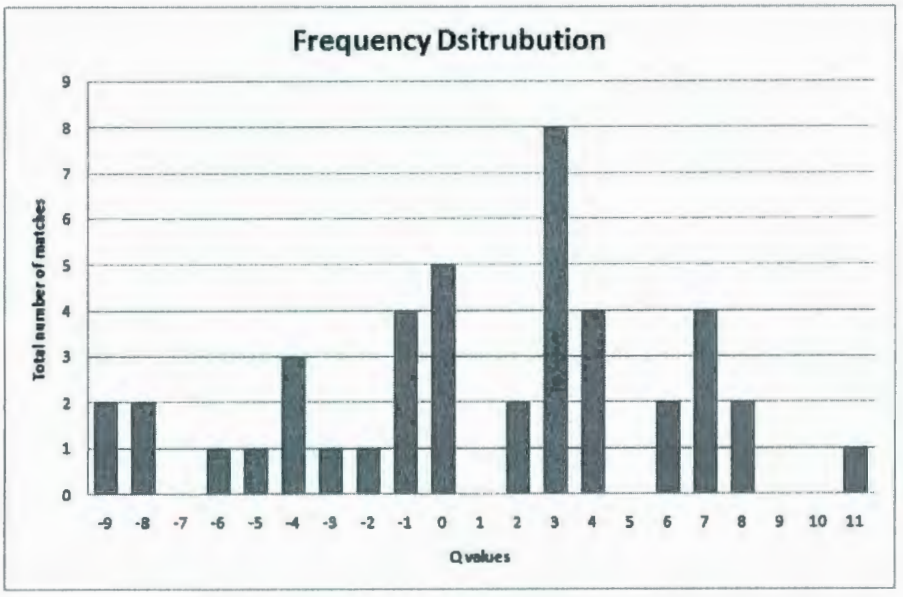


Figure 4.3: Frequency distribution of Q values.

Chapter 5

Combinatorial Pattern Matching

In general, a pattern is a theme of reoccurring events or elements of a set. In this thesis, by "patterns", we mean substrings of DNA sequences, repeating in some manner. When a pattern is seen in a DNA coding segment, biologists suspect that this may be an indication of functionality of a gene. Specifically, many patterns in DNA segments have been identified to be associated with diseases in humans. For example, the number of GAA repeats (the definition of repeats will be introduced in section 5.1) in an intron region of a gene is as large as 900. Such repeats of GAA in the human DNA is associated with the Friedreich Ataxia disease [12]. Around 50% of the human genome are repeats [49], so, the study of patterns in DNA sequences possesses a great deal of importance.

Given a collection of strings X, a database, and a string p, the pattern matching problem is to search for the presence of p in strings in X. In general, there are two types of pattern matching problems; exact and approximate. Given X and p, the exact pattern matching problem is to find substrings of strings in X that are exactly

the same as p; whereas approximate pattern matching is to find substrings of strings in X that equal to p with a few mismatches.

To find a pattern in thousands of genomes is computationally challenging. Using more sophisticated ways such as hash tables to organize data improves the search time from being impractical to somewhat practical. In this chapter, we find unique oligonucleotides (oligos) in strings of a database. Formal definition of unique oligos will be given later in the chapter. The search for unique oligos, a widely used application of string matching in biology, for the study of the functionality of cells, involves both exact and approximate pattern matching. We will look at some of the existing algorithms (see [49, 6, 35, 72]) for unique oligos searching. Based on these algorithms, we propose a parallelization technique in searching for the unique oligos to improve the search time. Furthermore, we propose a modified parallel algorithm based on those algorithms to improve the search time. References to this chapter are from [40, 72].

5.1 Repeats Searching

Let $x = AT\mathbf{ACCG}TAGT\mathbf{ACCG}TA\mathbf{ACCG}$ be a arbitrary sequence over the alphabet $\{A, C, G, T\}$. The substring $ACCG$ repeats three times in x. We call $ACCG$ a "repeat". There are numerous such repeats within any given genome and they are very important in genetic studies. In fact, repeats in DNA may provide clues for the tracing of evolution and understanding genetic diseases. Genomic Rearrangements[1] between organisms may indicate evolution through mutation and many genetic

[1]Genomic Rearrangements are differences in the order of genes, or in sequences within genes.

diseases including cancer are associated with a rearrangement of repeats within its genome[2].

In practice, we are interested in long repeats since they can provide more information about genes and mutations than shorter ones. However, it is clear that finding long repeats in a genome is more difficult and time consuming than finding short ones since finding long repeats require more comparisons. Thus, a simple approach to finding exact long repeats is to first find short q-mers for some small q and then extend them into longer l mers (ie. $q < l$) [49]. Recall that q-mer is a short substring of length q. Depending on different species, the meaningful values of l vary.

Example. Suppose we have the following sequence:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
A	A	C	G	T	A	G	T	A	T	C	G	T	A	G	T	A	T	G

The 4-mer $TAGT$ occurs at positions 4 to 7 and 12 to 15. Extending these 4-mers we can obtain two maximal repeats $GTAGTAT$, these are the 7-mers starting at positions 3 and 11.

Maximal repeats may always be detected in this manner by extending short repeats. Note that during the searching and extending process, the positions of l-mers must be identified and recorded in an organized fashion as the input database size may be very large.

[2]Genome is a collection of genes. This can be thought of as a database.

5.2 Exact Pattern Matching

As mentioned earlier, pattern matching is the process of finding all occurrences of a string p in strings of a database X.

Definition 1. Given a database $X = \{x_1, x_2, \ldots, x_k\}$ for some positive integer $k \geq 1$, we say a substring $q_{i,j}$ of length m occurs at jth position in x_i, where $m = |q| \leq |x_i|, 1 \leq i \leq k$.

Example. Let $X = \{ACCG, ACGTAG, TTTC\}$ and $q = TAG$. We say that q occurs at the fourth position of x_2. We may denote the substring TAG as $q_{2,4} = TAG$.

Algorithm 1: *Brute-Force algorithm for Pattern Matching*

Input: Pattern p and a database $X = \{x_1, x_2, \ldots, x_k\}$ for some positive integer $k \geq 1$ and $|p| \leq |x_i|$, where $1 \leq i \leq k$

Output: All locations of substrings of length m in X that match p.

```
begin
    m ⟵ length of p
    n_i ⟵ length of x_i
    for i ⟵ 1 to k do
        for t ⟵ 1 to (n_i − m + 1) do
            if q_(i,j) = p then
                output (i, j)
end
```

The simplest algorithm to find pattern p in a string of strings in X is shown in Algorithm 1. Let $q_{(i,j)}$ represent a substring of length m, where i refers to the ith

string in the database X and n_i is the size of the string x_i. If a substring $q_{(i,j)}$ equals p, then there is one occurrence of the pattern p in a string of strings in X. This is the brute-force algorithm that scans the entire strings of the database. At each position t, the program runs up to m operations for each string of strings in X, thus taking $O(kmn)$ time, the worst case time complexity.

5.3 Approximate Pattern Matching

In the exact pattern matching problem, matches for the exact pattern p are found in strings of the database being analyzed. However, due to the existence of mutations in DNA, it would be more sensible to investigate approximate pattern matches. For example, suppose that a gene mutation causes a DNA string $ACCG$ to become $ACCT$. In the exact pattern matching problem, these two strings are considered to be different. If for some positive integer d, two strings within d mismatches are said to be "approximately" the same. For instance, we say that $ACCG$ and $ACCT$ are approximately the same for $d = 1$. In the approximate pattern matching problem, we search for approximate matches of a pattern p with up to d mismatches, for some positive integer d.

In this section we are interested in finding unique oligos in a given database. Unique oligos are strings of certain length that appear only once in the database. Recall that a database is a collection of sequences. We find unique oligos by first identifying all substrings of some length from the database and comparing them with all the substrings of the same length in strings of the database. If the two substrings are the same, then they are not unique. We explore the idea of approximate pattern

matching using the Brute-Force method and two filtration methods. Using Brute-Force method, we generate all possible substrings of size l, ie., l-mers, from the database and compare them with other l-mers from the database to see which ones are unique. In the filtration methods, we search for all non-unique oligos in the database. Eliminating these non-unique oligos from the list of all possible l-mers results in the unique oligos of the database. To be unique, an oligo can appear any number of times exactly or approximately in one sequence of the database but the oligo cannot appear exactly or approximately in any other sequence [73]. For this reason, we do not compare with the other l-mers of the same sequence.

In this work, we will introduce the Brute-Force algorithm and two algorithms that use filtration method (see e.g., [49, 72]). Furthermore, we will implement these algorithms using Mathematica and parallelize them describing their improvements in search time, their time complexity, as well as suggesting a possible method to improve the search time.

5.3.1 Unique Oligonucleotides

In the past few years, several genomes including humans have been identified. The human genome contains about 3 billion base pairs [35]. Studying such a large set of data requires vast amount of computer resources. Many DNA analysis experiments involve breaking up the DNA to be investigated into short fragments, forming what is called a "DNA library".

A microarray typically is a solid substrate (e.g. glass or a semiconductor chip) on which spots of oligos have been deposited, each spot with a different known oligo. In

an experiment each DNA fragment in the DNA library attaches to a particular spot, ie. the olio which corresponds to it, so that many, if not all, of the DNA fragments are identified as well as the frequency of each kind of fragment. It is important that all the oligos used on the array are unique, so that the binding to each spot represents the expression of the corresponding gene.

In recent years, many biologists and computer scientists have been working on the unique oligo searching problem. Many algorithms have been developed, see [40, 49, 34, 35, 36, 6, 72, 73] for different sizes of the oligos. The size unique oligos were chosen differently for different species. For example, the length of unique oligos for Barley was chosen to be between 33 and 36 [72, 73] while for some bacteria species, the length was 25 [36].

We are interested in how to generate unique oligos from a collection of large sequences, by applying both the exact pattern matching problem and the approximate pattern matching problem.

Recall that given two strings x and y of the same length, $HD(x, y)$ denotes the hamming distance between x and y. If $HD(x, y) \leq d$, we say that x d-mismatches y or x is d-mutant of y.

Example. Suppose we have the following two sequences.

$$x: \ A \ C \ C \ G \ T \ T \ A \ G$$

$$y: \ A \ G \ G \ C \ T \ A \ A \ C$$

It follows that $HD(x, y) = 5$, and hence, x is d-mutant of y for any $d \geq 5$.

Recall that ESTs (Expressed Sequence Tags) are collection of DNA sequences. We think of ESTs as a database.

Definition 2. Given an ESTs set $X = \{x_1, x_2, \ldots, x_k\}$ for some k and d, a ***unique oligonucleotide*** is a substring of a string in the database with length l, which appears any number of times exactly in one EST sequence but does not appear exactly or approximately (within d mismatches) in any other EST sequence.

Example. Let $l = ACCGGT$, $d = 2$ and $X = \{x_1, x_2\}$. Suppose l occurs exactly in some position in x_1 and l occurs approximately (within 2 mismatch) in some position in x_2.

$$x_1 = \ldots \underbrace{ACCGGT}_{l} \ldots$$

$$x_2 = \ldots \underbrace{AC\dot{G}\dot{C}GT}_{l} \ldots$$

By the above definition, l is not unique.

Example. Suppose that l_1 and l_2 exist as substrings of different strings in X and $HD(l_1, l_2) \leq d$. Then l_1 is d-mutant of l_2 and itself. Similarly, l_2 is d-mutant of l_1 and itself. Thus, both l_1 and l_2 have 2 d-mutants (namely, l_1 and l_2), and hence, they are not unique.

5.3.2 Algorithms for the unique oligonucleotides design problem

Our purpose is to find all unique oligos occurring in a given database. We describe the Brute-Force method and the two filtration methods in the following. The references to this subsection are from [49, 72].

Method 1. The Brute-Force method.

By the definition of unique oligos, the simplest way to find all unique l-mers is to generate all substrings of size l from a string of strings in the database $X = \{x_1, x_2, \ldots, x_k\}$

first. Then we compare these l-mers with each other. From the comparison result, it is very easy to see which l-mers are unique. This simple and intuitive algorithm is called Brute-Force Method [36]. Recall n refers to the size of a database. As n becomes large, the time and space needed for Brute-Force Method becomes impractical. The time complexity for searching all l-mers in a length n of database is $\Theta(ln)$. Since there are four nucleotide bases in a DNA sequence, the number of comparisons between l-mers is $\frac{4^l(4^l-1)}{2}$. Then the total time complexity is $O(ln + \frac{4^l(4^l-1)}{2})$, which clearly illustrates that the Brute-Force method is very difficult for database containing millions and billions of bases.

Method 2. The two filtration methods.

The filtration method presents a faster technique to find unique oligos than the Brute-Force method. It may even be the fastest method so far, although to some extent, it is not as accurate as the Brute-Force method. The main point of the filtration technique is to find all non-unique l-mers from the database. Then, eliminating these non-unique l-mers from all the l-mers in the database, we can obtain unique l-mers. Some algorithms of the filtration technique have been introduced by [49, 35, 36, 72].

Algorithm 1. The basic l-mer filtration algorithm (see [49]) is based on the following observation. Suppose that two l-mers, l_1 and l_2 have d mismatches, i.e., $HD(l_1, l_2) \leq d$. If we divide both of them into $d+1$ substrings: $l_1^1 l_1^2 l_1^3 \dots l_1^{d+1}$, $l_2^1 l_2^2 l_2^3 \dots l_2^{d+1}$ and each l_j^i, except possibly l_j^{d+1} has length $\lceil \frac{l}{d+1} \rceil$, then there exists at least one $i_0 \in \{1, 2, \dots, d+1\}$, such that $l_1^{i_0} = l_2^{i_0}$. It is easy to see this by contradiction. Suppose that we cannot find any $i_0 \in \{1, 2, \dots, d+1\}$ such that $l_1^{i_0} = l_2^{i_0}$. Then there exists at least one mismatch between l_1^i and l_2^i for all $i \in \{1, 2, \dots, d+1\}$, and hence, there exist at least $d+1$ mismatches between l_1 and l_2. In practice, we

can choose suitable l and d such that $\frac{l}{d+1}$ is an integer. Let $q = \frac{l}{d+1}$. We say that if two l-mers have d-mismatches, then they share at least one identical q-mer. If an l-mer from EST x_1 is not unique, then we can find at least one l-mer from some EST sequences other than x_1 that are approximately the same. These two l-mers would share some q-mer. An effective way to find all candidates for non-unique l-mers is to find all q-mers that occur more than once and then extending them to l-mers.

This algorithm includes three steps. First, we find all strings of length q (i.e., q-mers) in a database X. The q-mers that occur more than once may be candidates to generate non-unique oligos. Then, we extend each identical q-mer into longer strings of length l and compare these l-mers with each other to identify their non-uniqueness. Recall that we do not compare two l-mers if they are from the same sequence in EST sequences. By an identical q-mer, we mean the same q-mer occurring in different EST sequences. Finally, we eliminate all non-unique l-mers from the total set of l-mers in X to obtain the unique ones. All the q-mers over the alphabet set $\{A, C, G, T\}$ are stored into a hash table. Each q-mer corresponds to an entry whose index is the hash value of the q-mer according to a hash function. This hash function assigns a unique value to each q-mer. Whenever a q-mer occurs in the database, its location gets recorded in the hash table. Once all the locations of all q-mers are found, we extend the often-occurring q-mers into l-mers and compare l-mers from different EST sequences.

Suppose that the total size of X is n. The time complexity for searching all q-mers is $\Theta(qn)$. The number of comparisons within each table entry is $O(r^2)$, where $r \approx n/4^q$ is the approximate number of identical q-mers in each table entry. Each comparison requires extension of the q-mers and takes $2(l-q)$ times. Noting that the hash table

has 4^q entries, we can finally obtain the overall time complexity $O((l-q)r^2 4^q)$ for the basic filtration method [72].

Algorithm 2. The algorithm in [72] is similar to the above basic filtration method, except that it is based on another observation. Suppose that two l-mers, l_1 and l_2 have d mismatches, i.e., $HD(l_1, l_2) \leq d$. If we divide both of them into $\lfloor \frac{d}{2} \rfloor + 1$ substrings: $l_1^1 l_1^2 l_1^3 \ldots l_1^{\lfloor \frac{d}{2} \rfloor + 1}, l_2^1 l_2^2 l_2^3 \ldots l_2^{\lfloor \frac{d}{2} \rfloor + 1}$ and each l_j^i, except possibly $l_j^{\lfloor \frac{d}{2} \rfloor + 1}$, has length $\lceil \frac{l}{\lfloor \frac{d}{2} \rfloor + 1} \rceil$, then there exists at least one $i_0 \in \{1, 2, \ldots, \lfloor \frac{d}{2} \rfloor + 1\}$, such that $l_1^{i_0}$ and $l_2^{i_0}$ have at most 1 mismatch, i.e., $HD(l_1^{i_0}, l_2^{i_0}) \leq 1$. It is also easy to see this by contradiction. If for any $i \in \{1, 2, \ldots, \lfloor \frac{d}{2} \rfloor + 1\}$, l_1^i and l_2^i have at least 2 mismatches, then the total mismatches between l_1 and l_2 would be at least $d+2$. In practice, one can also choose suitable l and d such that $\frac{l}{\lfloor \frac{d}{2} \rfloor + 1}$ is an integer. In this filtration algorithm, they also first search for all q-mers ($q = \frac{l}{\lfloor \frac{d}{2} \rfloor + 1}$), which they call seeds. Then before extending each identical q-mer to l-mers like we do for the basic filtration method, they cluster all the possible q-mers into groups such that within each group, every q-mer has no more than one mismatch with the other q-mers. Finally, for each group, they extend the q-mers to l-mers, compare those l-mers to see if they are non-unique, and then filter all non-unique l-mers from the total set of l-mers. The algorithm is implemented in C++ language.

Note that the idea of this filtration method is almost the same as the basic one, except for the clustering of 1 mutant seeds and extensions of 1 mutant seeds to l-mers and comparisons following, which results in $O(qr^2)$ comparisons, where $r \approx n/4^q$ and n is the size of the database. Thus, the overall time complexity is $O((l-q)q\frac{n^2}{4^q})$. By the applications of these two filtration algorithms to some database, we can see that algorithm 2 is actually faster than algorithm 1.

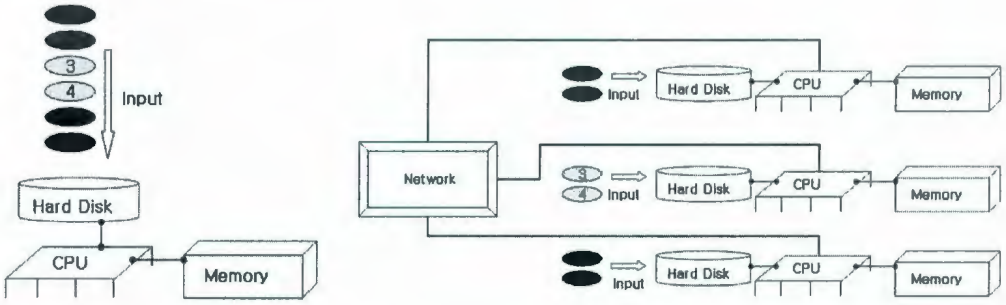
Remark 1. A q-mer that occurs in more than one EST sequence has to be extended to l-mers to the left and to the right positions for comparison. The extensions are done as follows. First, each identical q-mer is extended to the left most $l-q$ positions. For the next extension, the q-mer is extended to the left by $l-q-1$ positions and one position to the right. As the extension step proceeds, the number of possible extensions to the left decreases as the number of possible extension to the right increases. We can continue to extend until the number of possible extension to the left reaches zero and the number of possible extension to the right reaches $l-q$. For each identical q-mer, we can generate at most $(l-q+1)$ l-mers because if the q-mer is found at the very beginning of a sequence in a database, then it cannot be extended to the left position. This results in only one possible extension to the right by $l-q$ positions. A similar case happens when a q-mer to be extended is at the very end of a sequence.

5.3.3 Parallelization

To improve the existing algorithms for the unique oligos problem in a simple and feasible way, we think of the application of parallelization.

DNA analysis such as unique oligo design requires tremendous computational resources which include vast amounts of processing power and memory. Nowadays, we may not easily find any highly efficient single computers in universities or research institutions, but it is not hard at all to find many standard computers in any institution. Thus, the idea of parallel computing for DNA analysis is to let many computers share the load in order to get the job done faster.

As we know, a computer is composed of four main parts: central processing unit (CPU), memory, hard disk and user interface device. All the arithmetic operations are carried out in CPU. Memory is temporary storage to save the arithmetic operations and data. Inputs and outputs can be stored in the hard disk. A cluster is a group of computers that work together in a distributed memory system. In a distributed memory multiple-processor system, each processor has its own memory. This requires computational tasks to be distributed to the different processors for processing, making the system appropriate for designing parallel algorithms. The most common type of cluster is the *Beowulf* cluster (Figure 5.1b), computers connected with a TCP/IP Ethernet network.



(a) A general computation method with one cpu **(b)** A parallel computation method with multiple cpu's

Figure 5.1: a)A general computation method with one cpu. Six inputs must wait for their turns to be executed b)A parallelized computation method with three computers that take two inputs each. This is an example of Beowulf cluster.

Parallelization is an idea in which large problems can be divided into smaller ones that can be solved simultaneously. For implementing our parallel algorithms, we were

able to access over 40 *Beowulf* clusters through the departments of Mathematics and Statistics and Computer Science at Memorial University of Newfoundland.

For the Brute-Force method we have two steps: generation of l-mers and comparisons among l-mers. Given a database, we first need to generate all l-mers from all EST sequences of the database. The l-mers generation can be done in parallel as follows. Each processor is used to find all the l-mers in one sequence. The length of each sequence varies. Upon completion of the l-mers generation for some sequence, the processor receives the next sequence in line and repeats the task. Finally, all the detected l-mers from all the processors are combined in the main processor to have the total set of l-mers. For the comparison part, we pick one l-mer from one sequence and compare it with all the other l-mers in the other sequences to determine its uniqueness. Therefore, the parallelization idea is very simple. The algorithm for Brute-Force in parallel is given in Algorithms 2 and 3.

Algorithm 2: Brute-Force algorithm in parallel

Input: EST sequences $X = \{x_1, x_2, \cdots, x_n\}$

l: Length of unique oligonucleotides

d: Maximum number of mismatches for non-unique oligonucleotides

Output: All unique l-mers in X

begin

1. $lMers \leftarrow$ all l-mers generated from the database X ($lMer[[i]]$ contains all different l-mers from the sequence $X[[i]]$)

2. Comparisons for unique l-mers

$unilmers1 \leftarrow$ all unique l-mers (results of parallelization of map **goo[i]**)

3. Discard l-mers in $unilmers1$ which do not satisfy given conditions

$unilmers \leftarrow$ final results of unique l-mers.

end

Algorithm 3: The parallelizing portion of Brute-Force algorithm: each $lMer[[i]]$ will enter into the function for analysis on different computers.

goo[i]: **map to find all unique** l**-mers in** $X[[i]]$**:**

begin

$z \leftarrow$ a table to put all possible unique l-mers in $X[[i]]$

$lt \leftarrow$ a test table for uniqueness of l-mers in $X[[i]]$, initially assumed to be 1 for each component, i.e., each l-mer is assumed to be unique.

for *ii from* 1 *to length of* $lMers[[i]]$ **do**

  for *j from* 1 *to length of* X **do**

    for *jj from* 1 *to length of* $lMers[[j]]$ *with* $j \neq i$ **do**

      if $HammingDistance[lMers[[i, ii]], lMers[[j, jj]]] \leq d$ **then**

        $lt[[ii]] = 0$

$z \leftarrow$ all $lMers[[i, ii]]$ with $lt[[ii]] = 1$

Return[z]

end

The filtration method for the unique oligos searching problem also consists of two main parts: the generation of q-mers, the extension of q-mers and comparison of l-mers. The generation of q-mers is the same used in the previous method.

For the extension and comparison part, the parallelization idea depends on the specific algorithms. Using the filtration algorithm 1 without the application of parallelization requires first finding of all q-mers (or seeds) the occur more than once in the first sequence of EST sequences. Next, each seed is extended (l-mer) from the first sequence and compared with other extended l-mers in all the sequences except

the first one in the database to determine uniqueness. After this process, the next seed from the first sequence repeats the same procedure. In the whole process, each seed performs the extension and comparison independent of other seeds. We can parallelize by sending each seed into an available processor to perform the extension and comparison to determine uniqueness of each seed. After this step, each processor sends the recorded non-unique l-mers to the main processor and receives the next available seed. Finally, we find unique l-mers through filtration by the main processor. As for the filtration 2, it first clusters q-mers into groups such that in each group every q-mer is 1-mutant of each other. These q-mers are extended and compared, just like in the previous method. When we apply parallelization to this algorithm, we distribute all groups of seeds to different processors. Then each processor works on the extension and comparison for the related group of q-mers, and returns the recorded non-unique l-mers to the main processor later. The final filtration is also done by the main computer. The algorithms for the filtration methods in parallel is given in Algorithm 4 through Algorithm 7.

Algorithm 4: Parallelized Filtration algorithm 1

Input: EST sequences $X = \{x_1, x_2, \cdots, x_n\}$

l: Length of unique oligonucleotides

d: Maximum number of mismatches for non-unique oligonucleotides

$q = \frac{l}{d+1}$

Output: All unique l-mers in X

begin

1. $qMers \leftarrow$ all q-mers generated from the alphabet $\{A, C, G, T\}$

2. Extensions and comparisons for non-unique l-mers

$nonunipo \leftarrow$ positions of possible non-unique l-mers (results of parallelization of map **goo[qmer]**)

3. Filtration

$all \leftarrow$ all possible positions of l-mers in X

$uni \leftarrow$ all possible positions of unique l-mers ($Complement[all, nonunipo]$)

$unilmers1$: Apply "StringTake" to all elements in uni to obtain all unique l-mer candidates

4. Discard l-mers in $unilmers1$ which do not satisfy given conditions

$unilmers \leftarrow$ final results of unique l-mers.

end

Algorithm 5: The parallelizing portion of the algorithm 1: each q-mer will enter into this function for analysis on different computers.

goo[qmer]:

begin

$z \leftarrow$ a table for positions of non-unique l-mers resulted from extensions and comparisons for *qmer*

$Posi \leftarrow$ list of positions of *qmer* in X

(e.g., $Posi = \{\{1,2\},\{9,10\},\cdots,\{20,30\}\}$ corresponding to occurrences of *qmer*: $(1,1)$, $(1,2)$, $(2,9)$, $(2,10)$, $\cdots$, $(n,20)$, $(n,30)$, where (p_0,p_1) means position p_1 in sequence p_0.)

for j *from* 1 *to* $n-1$ **do**

  for jj *from* 1 *to* $Length[Posi[[j]]]$ **do**

    for k *from* $j+1$ *to* n **do**

      for kk *from* 1 *to* $Length[Posi[[k]]]$ **do**

        $q_1 \leftarrow$ the q-mer located at $(j, Posi[[j,jj]])$

        $q_2 \leftarrow$ the q-mer located at $(k, Posi[[k,kk]])$

        for *each pair of* l*-mers* l_1 *and* l_2 *that contain* q_1 *and* q_2, *respectively,* **do**

          if $HammingDistance[l_1,l_2] \leq d$ **then**

            $z = Append[z, \{p_0(l_1),p_1(l_1)\}, \{p_0(l_2),p_1(l_2)\}]$

            ($\{p_0(l_i),p_1(l_i)\}$ is the position of l_i)

  Return[z]

end

Algorithm 6: Parallelized Filtration algorithm 2

Input: EST sequences $X = \{x_1, x_2, \cdots, x_n\}$

l: Length of unique oligonucleotides

d: Maximum number of mismatches for non-unique oligonucleotides

$q = \frac{l}{\lfloor \frac{d}{2} \rfloor + 1}$

Output: All unique l-mers in X

begin

1. $qMers \leftarrow$ all q-mers generated from the alphabet $\{A, C, G, T\}$

$qp \leftarrow$ Groups of q-mers. Each entry of qp is a q-mer group in which each q-mer is one mutant of the other q-mers.

2. Extensions and comparisons for non-unique l-mers

$nonunipo \leftarrow$ positions of possible non-unique l-mers (results of parallelization of map **goo[qpi]**)

3. Filtration

$all \leftarrow$ all possible positions of l-mers in X

$uni \leftarrow$ all possible positions of unique l-mers ($Complement[all, nonunipo]$)

$unilmers1$: Apply "StringTake" to all elements in uni to obtain all unique l-mer candidates

4. Discard l-mers in $unilmers1$ which do not satisfy given conditions

$unilmers \leftarrow$ final results of unique l-mers.

end

Algorithm 7: The parallelizing portion of algorithm 2: each $qp[[i]]$ will enter into this function for analysis on different computers.

goo[qp[[i]]]:

begin

$z \leftarrow$ a table for positions of non-unique l-mers resulted from extensions and comparisons for q-mers in $qp[[i]]$

$Posi \leftarrow$ list of positions of all q-mers in $qp[[i]]$ in X

(e.g., $Posi = \{\{1,2\},\{9,10\},\cdots,\{20,30\}\}$ corresponding to occurrences of all q-mers in $qp[[i]]$: $(1,1)$, $(1,2)$, $(2,9)$, $(2,10)$, $\cdots$, $(n,20)$, $(n,30)$, where (p_0,p_1) means position p_1 in sequence p_0.)

for *j from* 1 *to* $n-1$ **do**

  for *jj from* 1 *to* $Length[Posi[[j]]]$ **do**

    for *k from* $j+1$ *to* n **do**

      for *kk from* 1 *to* $Length[Posi[[k]]]$ **do**

        $q_1 \leftarrow$ the q-mer located at $(j, Posi[[j,jj]])$

        $q_2 \leftarrow$ the q-mer located at $(k, Posi[[k,kk]])$

        for *each pair of* l*-mers* l_1 *and* l_2 *that contain* q_1 *and* q_2, *respectively,* **do**

          if $HammingDistance[l_1,l_2] \leq d$ **then**

            $z = Append[z, \{p_0(l_1),p_1(l_1)\}, \{p_0(l_2),p_1(l_2)\}]$

            ($\{p_0(l_i),p_1(l_i)\}$ is the position of l_i)

  Return[z]

end

We now analyze the time complexity for the parallelization for each algorithm. In general, the complexity for each algorithm has the following notion.

$$\frac{T(n)}{p} + C(n)$$

$T(n)$ defines the complexity of the algorithm (for n operations) we want to parallelize, where p is the number of processors in parallel computing. Thus, $\frac{T(n)}{p}$ is the complexity of a parallel algorithm. In parallel computing, there exists communication complexity C which is the communication time taken between a main computer and the processors used for parallelization. In detail, a main computer checks the availability of each processor. Once the main computer communicates with a processor that the processor is free, a task gets assigned for the processor. After successful completion of the task, the results get sent back to the main computer. These communication steps are determined experimentally, not analytically.

Suppose that the size of database X is n and p processors are used for parallelization. The time complexity for searching all q-mers in X by one computer is $\Theta(qn)$. Each processor works on n/p bases for the generation of q-mers. Zheng et al [72] assume that the average occurrences of a q-mer in X is r, where $r \approx n/4^q$. Each q-mer at some location in X has $(l-q+1)$ possible extensions to l-mers. For one type of extension to l-mers of a q-mer, there are r extensions and $\binom{r}{2}$ comparisons. Thus, one q-mer takes $(r+\binom{r}{2})(l-q+1)$ time for extension and comparison [72]. The total time for extension and comparison for all q-mers is then $(r+\binom{r}{2})(l-q+1)4^q$ if we do this in one computer. Since every processor does extensions and comparisons for $\frac{4^q}{p}$ q-mers. The overall time complexity for our basic filtration method is

$$\Theta(qn) + O((r+\binom{r}{2})(l-q+1)4^q) = \frac{O((l-q)r^2 4^q)}{p}.$$

With parallelization the complexity is

$$\Longrightarrow \frac{O((l-q)r^2 4^q)}{p} + C((l-q)r^2 4^q).$$

As for the parallelization for the filtration algorithm 2, the idea is almost the same as that of the basic one. There are 4^q possible q-mers of the alphabet $\{A, C, G, T\}$. We cluster q-mers such that each group contains at most 4 q-mers, along which each q-mer has 1-mismatch with the others. Every q-mer can be in q groups we may have $4^q \cdot q$ groups in total. Since every group has 4 elements, the number of different groups is $\frac{4^q q}{4} = q \cdot 4^{q-1}$. That is, we could have at most $q4^{q-1}$ groups of q-mers, so it takes $q4^{q-1}$ for clustering part. Then the extension and comparison for each group take $(4r + \binom{4r}{2})(l-q+1)$ time. Note that every processor works for $\lceil \frac{q4^{q-1}}{p} \rceil$ groups. Therefore, the overall time complexity for the filtration method in [72] is

$$\Theta(qn) + O(q4^{q-1}) + O((4r + \binom{4r}{2})(l-q+1)q4^{q-1}) = O(\frac{2q(l-q)r^2 4^q}{p}).$$

With parallelization the complexity becomes,

$$\frac{O(2q(l-q)r^2 4^q)}{p} + C(2q(l-q)r^2 4^q).$$

Similarly as we analyzed above, it is easy to see that for the Brute-Force algorithm working on a database with n bases and p processors, the time complexity for searching all l-mers is $\Theta(ln)$ and that for comparisons is $O(4^l(4^l-1))$. With parallelization the time complexity is,

$$\frac{O(4^l(4^l-1))}{p} + C(4^l(4^l-1)).$$

The above algorithms and their complexity analysis are based on the algorithms in [49, 6, 35, 72]. In fact, in the implementation of these algorithms, we improve

the filtration algorithms for the extension part. Recall that the filtration method 1 is based on the observation that if two l-mers are d-mutant to each other, then they share at least one q-mer provided that they are partitioned into $d+1$ q-mers (substrings of length q). Therefore, we should consider each q-mer that we generate from the database as one of the $d+1$ substrings of an l-mer. This indicates that when we extend a q-mer to an l-mer, we do not have to extend it base by base to the left or right. Instead, every time we extend the q-mer by kq (k is a positive integer such that $kq \leq l-q+1$) bases to the left or right. This results in at most d extensions to an l-mer for any q-mer in the database, rather than $l-q+1$ extensions as before. Thus, the time complexity for filtration method 1 becomes

$$\Theta(qn) + O((r + \binom{r}{2})d4^q) = O(dr^2 4^q).$$

With parallelization, the time complexity for filtration method 1 is

$$\frac{O(dr^2 4^q)}{p} + C(dr^2 4^q).$$

We can also apply this extension idea to filtration method 2 and obtain the time complexity as

$$\Theta(qn) + O(q4^{q-1}) + O((4r + \binom{4r}{2})dq4^{q-1}) = O(2qdr^2 4^q).$$

With parallelization the time complexity is,

$$\frac{O(2qdr^2 4^q)}{p} + C(2qdr^2 4^q).$$

As the parallelization idea has been determined, we decided to implement and parallelize the above mentioned algorithms using Mathematica, since it is a very

convenient software to deal with data, and in particular, it is very convenient to implement parallel algorithms.

Using built-in functions in Mathematica, we can easily generate a table for all possible 4^q q-mers, find the positions of all q-mers in the database, and distribute the q-mers into groups such that within each group each q-mer is one mutant of the other q-mers. For the filtration algorithms, we do not have a hash table to record all the occurrences of all q-mers before the extension. Instead, every time we pick one seed (i.e., one q-mer for the filtration algorithm 1 or one group of q-mers for the filtration algorithm 2), find all its positions in the database, and then extend it at different positions to l-mers for comparison. This takes less memory than the hash table. Moreover, we realize that the sets of all non-unique l-mers and all l-mers take large amounts of computer memory, so we just generate a table for all possible l-mer positions and only return the positions of non-unique l-mers from each processor to the main computer. Then, by filtration, we have all positions for unique l-mers and hence obtain all l-mers. In Mathematica programming, we can also conveniently use built in functions.

5.3.4 Test Runs

We wish to find unique oligos, which appear in only one sequence of a given set of genes. Our main focus is on the parallelization, that is, we illustrate that parallelized algorithms greatly reduce run time as compared to the serial ones.

We obtained the genomes of acaryochloris marina (173 genes with 128,904 bases), bacillus cereus (241 genes with 170,988 bases) and aspergillus nidulans (421 genes with

711,492 bases), which were taken from *National Center for Biotechnology Information* (http://www.ncbi.nlm.nih.gov.). These bacteria genomes were chosen arbitrarily for test purposes.

The main computer used for the parallel program has 1GB of memory and 2 GHz of CPU. Beowulf clusters have 4GB of Memory with 2.6GHz CPUs.

In our test runs, we are interested in unique oligos in these three databases. According to [44, 67, 6], in addition to being unique, the selected unique oligos have to fulfill the following conditions: the length of 25 nucleotides, includes at most 12 A or T nucleotides, includes at most 10 C or G nucleotides, includes at most 6 successive A, 6 successive T, 5 successive C or 5 successive G nucleotides, the GC content is $30\% - 70\%$, i.e., the total number of G and C nucleotides is between 8 and 17, no window of 8 nucleotides includes more than 6 A, 6 T, 4 C or 4 G nucleotides, and an inverse complementary nucleotide of an oligo can match at most 6 symbols from the beginning of an oligo.

Such conditions or parameters depend on different species. For barley, as presented in [72, 73], number of other parameters were required including GC content and melting temperatures. Parameters are related to the content and morphological shape of the probes used in microarray. The criteria for the parameters on long oligo probe can be found in [6].

Suppose that two 25-mers are approximately the same if they have 4 mismatches to each other (i.e., $d = 4$). According to the algorithms, $q = 5$ in the filtration algorithm 1 and $q = 8$ in the filtration algorithm 2. We implemented the Brute-Force method, the filtration algorithm 1 and the filtration algorithm 2 into Mathematica and also parallelized the three resulting programs. Then we ran the two filtration programs

in 1, 5, 10, 25, 44 processors, respectively. Brute-Force requires a great number of comparisons, thus, we parallelized Brute-Force method. Using 44 processors, we obtained the processing time for the parallelized Brute-Force method in Table 5.1. Table 5.2 shows the results of the processing time spent to obtain unique oligo with the filtration algorithm 1. On Table 5.3, the results of the processing time are recorded for the filtration method 2 and the Brute-Force method. Since the program for the Brute-Force method is very time consuming, we only ran it in 44 processors. The number of unique oligos for each methods can be found in Table 5.4.

organism	time
a. marina	4h45m17s
b. cereus	6h10m1s
a. nidulans	5d4h7m25s

Table 5.1: Processing time for parallelized Brute-Force method using 44 computers for the three test organisms, acaryochloris marina, bacillus cereus and aspergillus nidulans.

The processing times on Table 5.2 and Table 5.3 are plotted against the number of computer processors used on Figure 5.2 and Figure 5.3, respectively. We note two interesting observations. First, extensions by bases resulted in more comparisons. With increasing number of comparisons, we have more "chances" of obtaining the results close to Brute-Force method. However, when we applied the extensions method by $qMer$ size, it reduced the number of comparisons and obtained the unique oligos

much faster than the extension by bases method. Although the number of comparisons are decreased resulting in less accurate number of unique oligos, we have saved some time. Filtration method in the first place may not be the most accurate method, but as the database gets larger and larger for different organisms, this could be an efficient approach to solve the problem.

A second interesting observation is that effectiveness of our parallelization technique did not depend on the number of processors used as much as we expected. It would be sensible to think that the more processors you have the faster the program would be. However, from the graphs we note that the run time significantly reduces even after 5 processors from the single processor for aspergillus nidulans. For other organisms, the databases were too small to note any changes between different number of computers used. The main computer is responsible for assigning works to other processors and once each processor finishes its job, the main computer must effectively organize the data and process them accordingly. As the database size becomes larger, the returned values of non-unique oligos may become larger for each processor. Thus, when large amounts of data are returned to the main computer, and possibly at the same time, the main computer is unable to do its job at a given time, which may lead to delays.

From the database of acaryochloris marina with 173 genes of 128,904 bases, we find 48,999 different 8-mers in about 20 minutes. Noting that at most $4^8 = 65,536$ different 8-mers can be obtained from any database, we assume that for most of bigger databases, 65,536 different 8-mers will be obtained. Also, we suppose that almost all $4^5 = 1024$ different 5-mers can always be obtained from any test database. Thus, when we implement the filtration algorithms into Mathematica programs, we do not

generate all q-mers from the database. Instead, we directly generate all possible q-mers over the alphabet $\{A, C, G, T\}$ (which takes a split second) that resulted in the exact output as to first generating the q-mers from the database.

In Table 5.4, the number of unique oligos found using different methods can be found for the three test organisms. The number of unique oligos using the Brute-Force method is most reliable since each substring is compared with all others. The number of unique oligos for the filtration 2 when extended base by base includes a few more unique oligos compared to the ones by Brute-Force method. It is especially true for the filtration 2, qMer size extension. According to private communications with a local researcher[3] in biology and medicine that use microarray, these extra number of unique oligos is acceptable. The number of unique oligos using Brute-Force is desired. However, many unique oligos generated by manufactures may already contain errors such as those oligos that are not transcribed. Even with the best results, we may always have some errors contained in biological experiments. Filtration 2 using qMer size extension, performed in 3 minutes and 55 seconds, is an improvement over the Brute-Force method which took about a day with about 0.08% of extra unique oligos for aspergillus nidulans.

5.3.5 More Improvements

Other than parallelization, we propose more improvements for the old filtration methods to find the unique oligos in a given database.

Motivated by the old filtration algorithms, we can easily have some similar algorithms. One is based on the following observation. Suppose that two l-mers, l_1

[3]Dr. G Sun, Department of Medicine, Memorial University of Newfoundland

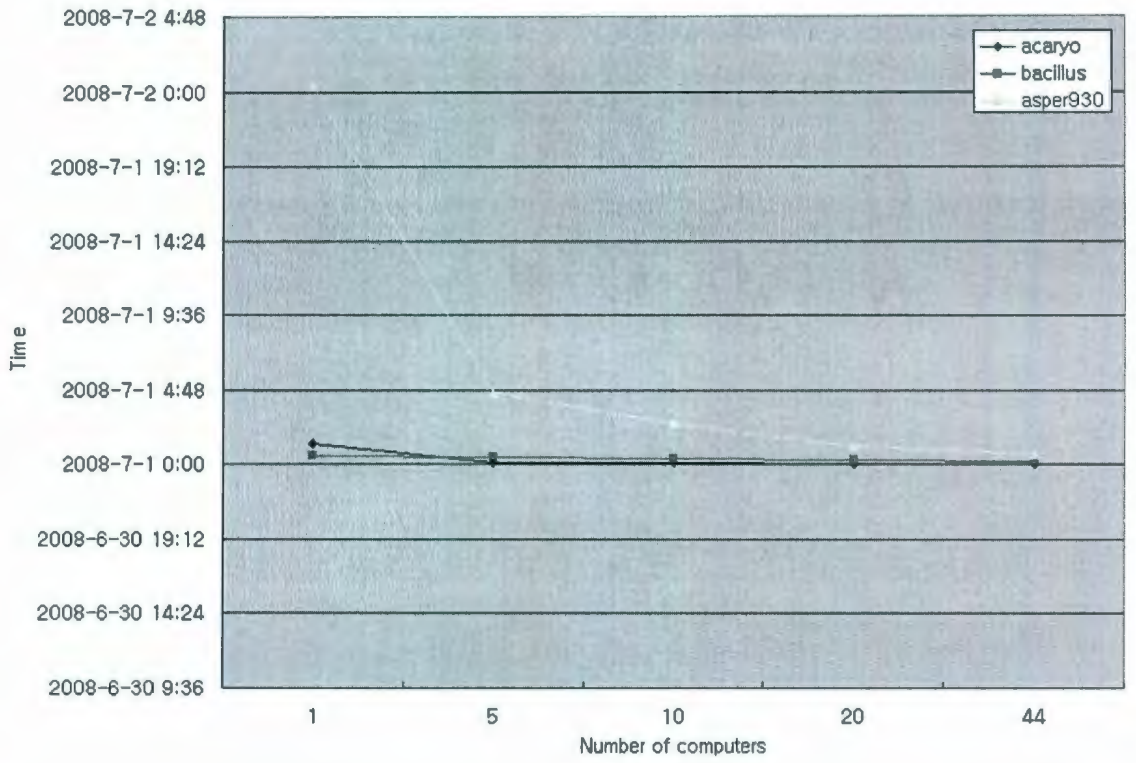


Figure 5.2: Estimating the run time of Algorithm 1 on different numbers of processors.

and l_2 have d mismatches, i.e., $HD(l_1, l_2) \leq d$. If we divide both of them into $\lfloor \frac{d}{3} \rfloor + 1$ substrings: $l_1^1 l_1^2 l_1^3 \ldots l_1^{\lfloor \frac{d}{3} \rfloor + 1}, l_2^1 l_2^2 l_2^3 \ldots l_2^{\lfloor \frac{d}{3} \rfloor + 1}$ and each l_j^i, except possibly $l_j^{\lfloor \frac{d}{3} \rfloor + 1}$, has length $\lceil \frac{l}{\lfloor \frac{d}{3} \rfloor + 1} \rceil$, then there exists at least one $i_0 \in \{1, 2, \ldots, \lfloor \frac{d}{3} \rfloor + 1\}$, such that $l_1^{i_0}$ and $l_2^{i_0}$ have at most 2 mismatches, i.e., $HD(l_1^{i_0}, l_2^{i_0}) \leq 2$. This is true since if for any $i \in \{1, 2, \ldots, \lfloor \frac{d}{3} \rfloor + 1\}$, l_1^i and l_2^i have at least 3 mismatches, then the total mismatches between l_1 and l_2 would be at least $d + 3$. Based on this observation, we can have a new algorithm. First find all q-mers ($q = \lceil \frac{l}{\lfloor \frac{d}{3} \rfloor + 1} \rceil$) in the database and cluster them into groups such that within each group, each q-mer has no more than 2 mismatches with the other q-mers. Then extend the q-mers in each group to l-mers and compare

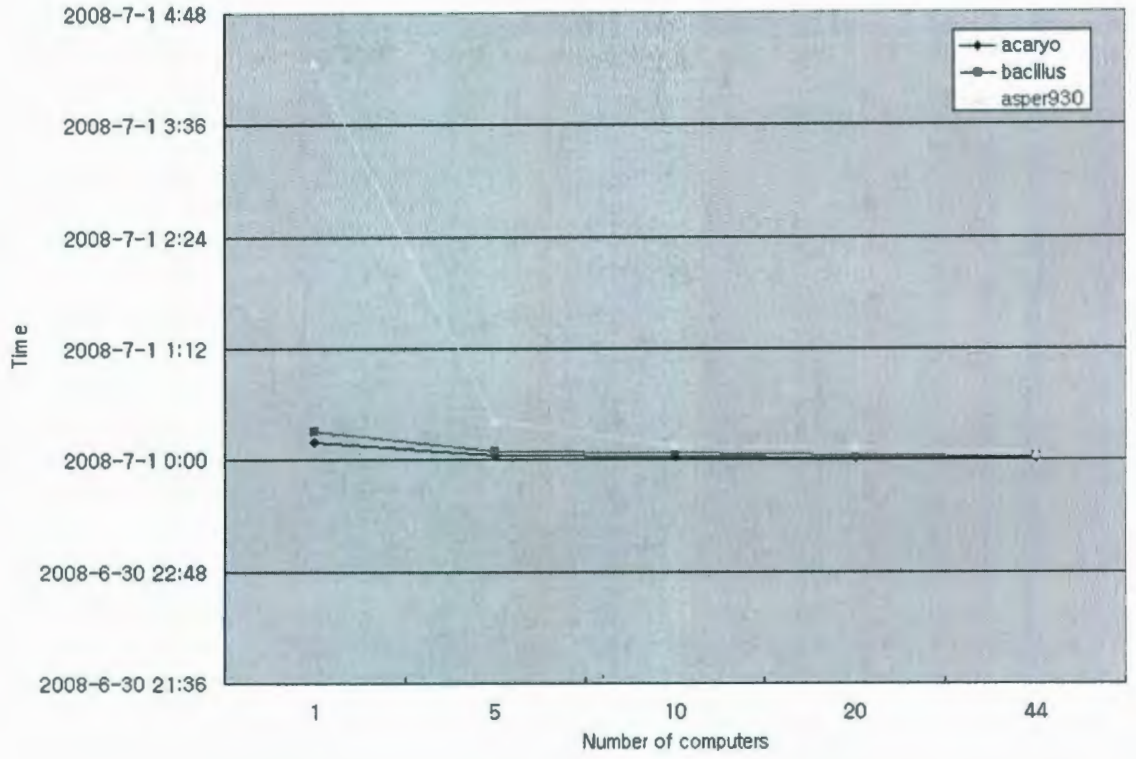


Figure 5.3: Estimating the run time of algorithm 2 with the 1-mutant on different numbers of processors.

the resulted l-mers to see if they are non-unique, and finally eliminate the non-unique l-mers from the total l-mers to obtain the unique l-mers. The detailed steps of the algorithm are as follows.

(i) (1) Find all q-mers ($q = \lceil \frac{l}{\lfloor \frac{d}{3} \rfloor + 1} \rceil$) in the database X.

(2) Make a table "qmergroups" such that in each table entry, a group of q-mers are recorded and each q-mer has no more than 2 mismatches with the other q-mers.

(ii) (1) For each entry of "qmergroups", find all positions of q-mers in this entry

and record these positions (sequence by sequence) into a table "qmerpositions".

(2) Fix one position from one (i-th) entry of "qmerpositions", say L_1.

(3) Choose a position from another entry of "qmerpositions", say L_2. The q-mers at L_1 and L_2 are called q_1 and q_2, respectively, for convenience. Extend q_1 and q_2 correspondingly to $l-$mers. Compare the two $l-$mers to see whether the Hamming Distance between them is less than d. If so, these $l-$mers are not unique oligo.

(4) Choose all the other possible positions from entries other than the i-th entry of "qmerpositions" and repeat (3).

(5) Fix another position from the i-th entry of "qmerpositions" and repeat (3)-(4). Repeat this step till all positions from the i-th entry of "qmerpositions" have been considered.

(6) Choose another entry of "qmerpositions" and repeat (2)-(5). Repeat this step till all entries of "qmerpositions" have been considered.

(7) Choose another entry of "qmergroups" and repeat (1)-(6). Repeat this step till all entries of "qmergroups" have been considered.

Actually, we did not implement this algorithm into a program. We think it should be interesting and maybe will work on this as a future effort. However, following this idea, we may have a few algorithms based on similar observations and they may be not bad algorithms from some points of view. In this series of algorithms, the number q, the length of small strings which are candidates for extension, is increasing from $\lceil \frac{l}{d+1} \rceil$ to $\lceil \frac{l}{\lfloor \frac{d}{2} \rfloor + 1} \rceil$, $\lceil \frac{l}{\lfloor \frac{d}{3} \rfloor + 1} \rceil$, and finally to $\lceil \frac{l}{2} \rceil$ and even l. Thus, the size of possible q-mers, 4^q, is also becoming larger and larger. This may result in impossible algorithms

for current computers because of memory limitation or other problems. We do not know what would happen. This also seems to be an interesting problem.

			Organisms		
			a. marina	b. cereus	a. nidulans
		no. genes	173	241	421
		nt length	128,904	170,988	711, 492
Processing time for filtration method 1 using various numbers of computers	serial	1	1h17m52s (5h27m38s)	32m49s (6h10m53s)	24h24m26s (7d14h19m38s)
	Parallel	5	7m7s (30m8s)	29m22s (1h53m20s)	4h37m56s (21h44m4s)
		10	4m24s (19m49s)	20m15s (1h38m13s)	2h36m54s (11h11m34s)
		20	3m30s (16m33s)	17m0s (1h6m14s)	1h10m35s (5h37m32s)
		44	3m30s (8m30s)	11m7s (51m42s)	40m17s (3h43m29s)

Table 5.2: Processing time table for filtration method 1 for three test organisms, acaryochloris marina, bacillus cereus and aspergillus nidulans. The time in the non-bracket is extension by qMer size and the time in bracket is the extension by bases. Extension by bases produces more comparisons, thus lengthening the processing time.

			Organisms		
			a. marina	b. cereus	a. nidulans
Processing time for filtration method 2 using various numbers of computers	serial	1	11m50s (22m0s)	18m17s (1h20m46s)	4h3m48s (14h20m17s)
	Parallel	5	3m14s (4m22s)	5m19s (10m56s)	25m12s (1h20m35s)
		10	1m57s (3m37s)	4m58s (10m20s)	8m30s (46m44s)
		20	56s (1m42s)	2m59s (5m27s)	6m55s (20m4s)
		44	47s (59s)	1m55s (4m35s)	3m55s (16m2s)

Table 5.3: Processing time table for filtration method 2 for the three test organisms. The time in the non-bracket is extension by $qMer$ size and the time in bracket is the extension by bases.

Methods	Organisms		
	a. marina	b. cereus	a. nidulans
Brute-Force	119,448	162,700	695,115
filtration 1 (base by base)	119,448	162,700	695,115
filtration 2 (base by base)	119,486	162,735	695,210
filtration 1 (qMer size extension)	119,448	162,700	695,115
filtration 2 (qMer size extension)	119,645	162,999	695,668

Table 5.4: Number of unique oligos found using different methods for the three test organisms.

Chapter 6

Conclusions

There are several aspects of research in DNA analysis. This thesis is an exploration of four different areas of DNA analysis that use Combinatorics and its applications. Gavin et al [26] applied the idea of Levenshtein distance and created large sets of synthetic tissue identification. The identification tags provided error detection and correction. Gavin et al[26] used the error detection and correction capability of up to only two distances. Error-detecting and error-correcting codes of various lengths can be constructed using design theory. In Design Theory, codes capable of correcting errors of lengths up to seven have been identified. Gavin et al[26] applied the Levenshtein idea to improve the accurate identification of tissue source in the presence of errors even though their paper only considered the ability to detect and correct up to two substitution errors.

The second area of DNA analysis using Combinatorics is the application of Graph Theory. We studied two methods of sequencing technique, fragmentation (overlap) method and sequencing by hybridization (SBH), both of which use Graph Theory.

These sequencing techniques are very important in the advancement of biological sciences. Fragmentation method uses each base position of DNA chain to be determined individually (eg. gel electrophoresis). On the other hand, SBH uses sets of oligos to be hybridized, thus, allowing analysis of DNA samples up to several kilobases which are much longer than the samples produced by fragmentation method. Graph theory and combinatorial algorithms play a key role in understanding and performing SBH reconstruction [2]. SBH method still needs to be studied and improved since it can result in multiple "possible" DNA sequences [51].

Third area of DNA analysis that we studied was sequence comparison. Whenever there may be an error in gene expression, one may inspect the normal expressed sequences and the one with errors. It is often impossible to suspect whether the gene expression contains errors or not after a specific stage in the expression. Errors in gene expression may be detected in much later stages as the cell with errors become suspicious as with those cancerous cells. Then, we backtrack the stages to study what has gone wrong in the expression. DNA sequences are expressed in terms of four nucleotides whereas protein sequences are expressed by 20 different amino acids. Two sequences are aligned (paired) element by element. Locating different positions of alignment, we can identify what options give us the best alignment by the scores of these options. Dynamic programming is used to effectively pair up two sequences. Heuristic searches including FASTA are still being developed to improve the search time. The different sequences from the same cell do not mean that there is an error. Rather, it means that some genes are activated to express while others are not.

The final area of DNA analysis studied is the efficient selection of unique nucleotide from a database containing large DNA or protein sequences. Even though technology

has improved significantly over the past decade, it is not sufficient to handle an ever increasing production of large data. Technology such as the microarray works by exposing a given DNA molecule to hybridize to its complementary DNA template requiring unique oligos of certain length. This length varies for different species. Intuitively, to find all unique oligos in a DNA data set, we can find all oligos of the given length and compare them with each other. However, while the speed and the memory capacity of a computer improved significantly, it is still impossible to generate all oligos of large length from a large database. For example, barley, with the current data size of 56 Mb, requires unique oligos of length about 36 [73], which means that it requires up to $4^36 \simeq 4.7 \times 10^{21}$ bytes of computer memory. This is impossible for current computers. Analysis of such large data require an effective approach such as those given in [35, 40, 72]. In this thesis, we studied the Brute-Force method, which is the intuitive method mentioned above, and the filtration methods for the selection of unique oligos. In particular, we improved the existing filtration methods to an extent, and implemented them in Mathematica. We then applied paralleled these algorithms. We used a small example to explain the technique of parallelization (see Appendix). We discussed the time complexity for the algorithms we used as well as the modified-parallelized algorithms. By the simulations for DNA databases of some species (acaryochloris marina, bacillus cereus and aspergillus nidulans), we saw that, even with 5 processors, parallelization improve the time significantly. However, as the number of processors increase there was no need of extra processors. This interesting factor was due to the communication complexity that took place between the main computer and each processor. Machines that does not use or limit the use of communication factors are becoming popular these days. Such machines have

multiple processors in one computer (ie. duo and quad processors) which significantly improves the communication time between the processors. We also proposed more improvements that can be done to the algorithms we have mentioned. Parallelization techniques provide much faster way for DNA analysis (including the selection of unique oligos) and they make it more effective to do analysis for large DNA databases with available computers.

Bibliography

[1] Adams MD, Kelly JM, Gocayne JD, Dubnick M, Polymeropoulos MH, Xiao H, Merril CR, Wu A, Olde B, Moreno RF, Kerlavage AR, McCombie WR and Venter JC "Complementary DNA sequencing: expressed sequence tags and human genome project" *Science* 252 (1991) pp. 1651-1656.

[2] Adams P, Bryant D and Byrnes S "Application of Graph Theory in DNA sequencing by hybridization" *Bulletin of the Institute of Combinatorics and its Applications* 31 (2001) pp. 13-20.

[3] Anderson I "Combinatorial desigms and Tournaments" *Oxford Science Publications, Clarendon Press, Oxford* (1997).

[4] Altschul SF, Gish W, Miller W, Myers EW and Lipman DJ "Basic local alignment search tool" *Journal of Molecular Biology* 215 (1990) pp. 403-410.

[5] Altschul SF, Madden TL, Schaffer AA, Zhang J, Zhang Z, Miller W and Lipman DJ "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs" *Nucleic Acid Research* 25-17 (1997) pp. 3389-3402.

[6] Aluru S "Handbook of Computational Molecular Biology" *Chapman and Hall CRC* (2006) pp. 6-19, 13-1, 13-2, 24-1, 24-3.

[7] Assaf A, Shalaby N and Yin J "Directed packing and covering designs with block size of four" *Discrete Mathematics* 238 (2001) pp. 3-17.

[8] Bains W and Smith G"A novel method for nucleic acid sequence determination" *Journal of Theoretical Biology* 135 (1988) pp. 303-307.

[9] Barlow RJ "Statistics: A guide to the use of statistical methods in the physical sciences" *Wiley* (1989) pp. 24-33.

[10] Bergeron B "Bioinformatics Computing" *Pearson Education Inc* (2003)

[11] Bours PAH "On the construction of Perfect Deletion-Correcting Codes using Design Theory" *Designs, Codes and Cryptography* 6-1 (1995) pp. 5-20.

[12] Campuzano V, Montermini L, Molto' MD, Pianese L, Cosse'e M, Cavalcanti F, Monros E, Rodius F, Duclos F, Monticelli A, Zara F, Can izares J, Koutnikova H, Bidichandani SI, Gellera C, Brice A, Trouillas P, De Michele G, Filla A, De Frutos R, Palau F, Patel PI, Di Donato S, Mandel JL, Cocozza S, Koenig M and Pandolfo M "Friedreich's ataxia: autosomal recessive disease caused by an intronic GAA triplet repeat expansion" *Science* Mar 8,271-5254 (1996) pp. 1374-1375.

[13] Chafe K "Applications of Graph Theory to DNA and RNA Sequencing" *Memorial University of Newfoundland* Department of Mathematics and Statistics, Honours Thesis (2004).

[14] Chen X, Kwong S and Li M "A compression algorithm for DNA sequences and its application in genome comparisons" *Genome Informatics* 10 (1999) pp. 51-61.

[15] Colbourn CJ and Dinitz JH "Mutually orthogonal latin squares: a brief survey of constructions" *Journal of Statistical Planning and Inference* 95-1-2 May (2001) pp. 9-48.

[16] Cormen TH, Leiserson CE, Rivest RL and Stein C "Introduction to Algorithms-Second Edition" *The MIT Press* Cambridge, Massachusetts (2001) pp. 10,41-45.

[17] Cover TM, Thomas JA "Elements of Information Theory" *Wiley-IEEE* (2006) pp. 212.

[18] Colbourn CJ and Dinitz JH "The CRC Handbook of Combinatorial Designs" *CRC Press Inc., Boca Raton* Second Edition (2006).

[19] Crick F "Central Dogma of Molecular Biology" *Nature* 227 (1970) pp. 561-563.

[20] Croce CM "Oncogenes and Cancer" *The New England Journal of Medicine* 358-5 January 31 (2008) pp. 502-511.

[21] Davies H, Bignell GR, Cox C, Stephens P, Edkins S, Clegg S, Teague J, Woffendin H, Garnett MJ, Bottomley W, Davis N, Dicks E, Ewing R, Floyd Y, Gray K, Hall S, Hawes R, Hughes J, Kosmidou V, Menzies A, Mould C, Parker A, Stevens C, Watt S, Hooper S, Wilson R, Jayatilake H, Gusterson BA, Cooper C, Shipley J, Hargrave D, Pritchard-Jones K, Maitland N, Chenevix-Trench G, Riggins GJ, Bigner DD, Palmieri G, Cossu A, Flanagan A, Nicholson A, Ho JW, Leung SY, Yuen ST, Weber BL, Seigler HF, Darrow TL, Paterson H, Marais R, Marshall CJ, Wooster R, Stratton MR, and Futreal PA "Mutations of the BRAF gene in human cancer" *Nature* 27-417 June (2002) pp. 949-954.

[22] Dembo A, Karlin S and Zeitouni O "Limit distribution of maximal non-aligned two-sequence segmental score" *Ann. Prob* 22 (1994) pp. 2022-2039.

[23] Dori S and Landau GM "Construction of Aho Corasick Automaton in Linear Time for Integer Alphabets" *Information Processing Letters* 98-2 (2006) pp. 66-72.

[24] Drmanac R, Drmanac S, Chui G, Diaz R, Hou A, Jin H, Jin P, Kwon S, Lacy S, Moeur B, Shafto J, Swanson D, Ukrainczyk T, Xu C, Little D "Sequencing by hybridization (SBH): advantages, achievements, and opportunities" *Adv Biochem Eng Biotechnol* 77 (2002) pp. 75-101.

[25] Franca LTC, Carrilho E and Kist TBL "A review of DNA sequencing techniques" *Quarterly Reviews of Biophysics* 35-2 (2002) pp. 169-200.

[26] Gavin AJ, Scheetz TE, Roberts CA, O'Leary B, Braun TA, Sheffield VC, Soares MB, Robinson JP and Casavant TL "Pooled Library tissue tags for EST-based gene discovery" *Bioinformatics* 18-9 (2002) pp. 1162-1166.

[27] Gibas C and Jambeck P "Developing Bioinformatics Computer Skills" *O'Reilly* (2001)

[28] Goodaire EG and Parmenter MM "Discrete Mathematics with Graph Theory" *Prentice Hall* Third Edition (2006)

[29] Griffith AJF, Miller JH, Suzuki DT, Lewontin RC and Gelbart WM "An Introduction to Genetic Analysis" *Freeman, New York* (2000) pp.2-10.

[30] Guruswami V "List Decoding of Error-Correcting Codes" *Lecture Notes in Computer Science* 3282 (2004) pp.1-14.

[31] Hoffman DG, Leonard DA, Lindner CC, Phelps KT Rodger CA and Wall JR "Coding Theory: The Essentials" *Marcel Dekker Inc. New York* (1991)

[32] Hood L and Galas D "The digital Code of DNA" *Nature* 421-6921 (2000) pp. 444-448.

[33] Hudson D "Lecture Notes: Algorithms in Bioinformatics I" *ZBIT:Zentrum fur Bioinformatik Tubingen* (2006) pp. 3-78.

[34] Hyyro H "On applying string matching in searching unique oligonucleotides" *Proceedings of the 2001 International Conference on Mathematics and Engineering Techniques in Medicine and Biological Sciences* CS Press (2001) pp. 1-7.

[35] Hyyro H, Juhola M and Vihinen M "On exact string matching of unique oligonucleotides" *Computers in Biology and Medicine* 35 (2005) pp. 173-181.

[36] Hyyro H, Vihinen M and Juhola M "On Approximate string matching of unique oligonucleotides" *Medinfo* 10 (2001) pp. 960-964.

[37] Jonassen I "Efficient discovery of conserved patterns using a pattern graph" *CABIOS* 13-5 (1997) pp. 50-522.

[38] Karlin, S and Altschul SF "Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes" *Proc. Natl. Acad. Sci. USA* 87 (1990) pp. 2264-2268

[39] Korf I, Yandell M and Bedell J"BLAST: An Essential Guide to the Basic Local Alignment Search Tool" *O'Reilly* (2003) pp. 100-101.

[40] Jones NC and Pevzner PA "An Introduction to Bioinformatics Algorithms" *MIT Press, London, England* (2004)

[41] Kurtz S and Schleiermacher C "REPuter: fast computation of maximal repeats in complete genomes" *Bioinformatics: Applications Note* 15-5 (1999) pp.426-427.

[42] Keller GH and Manak MM "DNA Probes" *M Stockton Press* Second Edition (1993) pp.1-21.

[43] Levenshtein VI "On Perfect Codes in deletion and insertion metric" *Discrete Mathematics Appl.* 2-3 (1992) pp. 241-258.

[44] Lockhart DJ, Dong H, Byrne MC, Follettie MT, Gallo MV, Chee MS, Mittmann M, Wang C, Kobayashi M, Horton H and Brown EL "Expression monitoring by hybridization to high-density oligonucleotide arrays" *Nature Biotechnology.* 14 (1996) pp. 1675-1680.

[45] Lysov Y, Florent'ev V, Khorlin A, Khrapko K, Shik V and Mirzabekov A "DNA sequencing by hybridization with oligonucleotides" *Koklady Academy Nauk USSR.* 303 (1988) pp. 1508-1511.

[46] Mahmoodi A "Existence of Perfect 3-Deletion-Correcting Codes" *Designs, Codes and Cryptography* 14 (1998) pp. 81-87.

[47] Nagaraj SH, Gasser RB and Ranganathan S "A hitchhiker's guide to expressed sequence tag (EST) analysis" *Briefings in Bioinformatics* 8-1 (2006) pp. 6-21.

[48] Parida L "Pattern Discovery in Bioinformatics" *Chapman and Halls/CRC* (2008) pp.89-106.

[49] Pevzner PA "Computational Molecular Biology: An Algorithmic Approach" *MIT Press, London, England* (2000)

[50] Pevzner PA "DNA physical mapping and alternating EUlerian cycles in coloured graphs" *Algorithmica* 13 (1995) pp. 77-105.

[51] Pevzner PA and Sze SH "Combinatorial approaches to finding subtle signals in DNA sequences." *In: Proceedings of the First IEEE Computer Society Bioinformatics Conference (CSB'02)* IEEE Press (2002)

[52] Rahmann S "Rapid large-scale oligonucleotide selection for microarrays" *In: Proceedings of the International Conference on Intelligent Systems for Molecular Biology* AAAI press, Menlo Park, CA (2000) pp. 269-278.

[53] Sanger F, Nicklen S and Coulson AR "DNA sequencing with chain-terminating inhibitors" *Proceedings of the National Academy of Sciences of the United States of America* 74 (1977) pp. 5463-5467.

[54] Schena M "Microarray Analysis" *John Wiley and Sons Inc* (2003) pp. 1-25, 121-157.

[55] Shalaby N, Wang J and Yin J "Existence of Perfect 4-Deletion-Correcting Codes with length six" *Designs, Codes and Cryptography* 27-1,2 (2002) pp. 145-156.

[56] Siede W, Kow Y W, and Doetsch P W "DNA Damage Recognition" *Taylor and Francis, New York.* (2006)

[57] Smith TF and Waterman MS "Identification of Common Molecular Subsequences" *Journal of Molecular Biology.* 147 (1981) 195-197.

[58] Stekel D "Microarray Bioinformatics" *Cambridge University Press.* (2003) pp.43-61

[59] Street DJ and Seberry J "All DBIBDs with block size four exist" *Utilitas Mathematica* 18 (1980) pp. 27-34.

[60] Voet D, Voet JG and Pratt CW "Fundamentals of Biochemistry" *John Wiley and Sons Inc., New York* (1999)

[61] Wang J, and Yin J "Construction for Perfect 5-Deletion-Correcting Codes of length seven" *IEEE Transations on Informations Thoery* 52-8 (2006) pp. 3676-3685.

[62] Watson, J "Molecular Structure of Nucleic Acid: A Structure for Deoxyribose Nucleic Acid" *American Journal of Psychiatry* 160-4 April (2003) pp. 623-624.

[63] Watson, JD and Crick FHC "A Structure for Deoxyribose Nucleic Acid" *Nature* 171 (1953) pp. 737-738.

[64] Wesselink JJ, Iglesia B, James SA, Dicks JL, Roberts IN and Rayward-Smith VJ "Determining a unique defining DNA sequence for yeast species using hashing techniques" *Bioinformatics* 18-7 (2002) pp. 1004-1010.

[65] Wikipedia "www.wikipedia.org" *1* 21-3 March (2003) pp. 497-503.

[66] Wilson, RM "An existence theory for pairwise balanced designs" *J. Combinatorial Theory* 13A (1972) pp. 220-245.

[67] Wodicka L, Dong H, Mittmann M, Ho M and Lockhart DJ "Genome-wide expression monitoring in Saccharomyces cerevisiae" *Nature Biotechnology* 15 (1997) pp. 1359-1367.

[68] Xia X "Bioinformatics and the Cell: Modern Computational Approaches in Genomics, Proteomics and Transcriptomics" *Springer USA* (2007) pp. 1-22.

[69] Yin J "A Combinatorial Construction for Perfect Deletion-Correcting Codes" *Designs, Codes and Cryptography* 23 (2001) pp. 99-110.

[70] Yin J "Existence of directed GDDs with block size five and index $\lambda \geq 2$" *Journal of Statistical Planning and Inference* 86 (2000) pp. 619-627.

[71] Yoota J "Tumor progression and metastasis" *Carcinogenesis* 21-3 March (2003) pp. 497-503.

[72] Zheng J, Close TJ, Jiang T and Lonardi S "Efficient Selection of Unique and Popular Oligos for Large EST Databases" *Bioinformatics* 20-13 (2004) pp. 2101-2112.

[73] Zheng J, Svensson JT, Madishetty K, Close TJ, Jiang T and Lonardi S "OligoSpawn: a software tool for the design of overgo probes from large unigene datasets" *BMC Bioinformatics* Jan 7-7 (2006) pp. 1-9

Appendix A

Algorithms

Algorithm 8: Needleman-Wunsch algorithm

Input: Two sequences, S and T

Output: Optimal alighment and score δ

Initialization: Set $d[i,0] = -i\gamma$ for $i = 0, 1, \ldots, m$

Set $d[0,j] = -j\gamma$ for $j = 0, 1, \ldots, n$

begin

for $i = 1, 2, \ldots, m$ **do**

For $i = 1, 2, \ldots, m$ Set

$$d[i,j] = max \begin{cases} d[i,j-1] - \gamma \text{ the gap in sequence X} \\ d[i-1,j-1] + p(i,j) \\ d[i-1,j] - \gamma \text{ the gap in sequence Y} \end{cases}$$

Set backtrack B[i,j] to the maximizing pair

The score is $\delta = d[m,n]$

Set $[i,j] = [m,n]$

repeat

if $B[i,j] = [i-1, j-1]$ **then**

Print S_i and T_j

else

if $B[i,j] = [i-1, j]$ **then**

Print S_i

Print T_j

Set [i,j]=B[i,j]

until $[i,j] = [0,0]$

end

Algorithm 9: Smith-Waterman algorithm

Input: Two sequences, S and T

Output: Optimal alighment and score δ

Initialization: Set $d[i,0] = -i\gamma$ for $i = 0, 1, \ldots, m$

Set $d[0,j] = -j\gamma$ for $j = 0, 1, \ldots, n$

begin

  for $i = 1, 2, \ldots, m$ **do**

    For $i = 1, 2, \ldots, m$ Set $d[i,j] = max \begin{cases} d[i,j-1] - \gamma \\ d[i-1,j-1] + p(i,j) \\ d[i-1,j] - \gamma \\ 0 \end{cases}$

    Set backtrack B[i,j] to the maximizing pair

  Set $[i,j] = max\{d[i,j]$ for $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, n\}$

  The score is $\delta = d[i,j]$

  repeat

    if $B[i,j] = [i-1, j-1]$ **then**

      Print S_{i-1} and T_{j-1}

    else

      if $B[i,j] = [i-1, j]$ **then**

        Print S_{i-1}

      Print T_{j-1}

    Set [i,j]=B[i,j]

  until $d[i,j] = 0$

end

Appendix B

Small Example

Suppose we have a database $X = \{x_1, x_2, x_3\}$, where

$$x_1 = ACCACGCT,$$
$$x_2 = GGACGCTGC,$$
$$x_3 = GCGCTGCAC.$$

We search for unique oligos of length $l = 6$. Assume that two l-mers are approximately the same if the Hamming Distance between them is at most 2, i.e., $d = 2$. Let $l_{i,j}$ refer to the l-mer located at the jth position in sequence x_i of X. For example, $l_{1,2} = CCACGC$ occurs at the second position of x_1.

For the Brute Force Method, we generate all possible l-mers in X:

$$l_{1,1} = ACCACG,\ l_{1,2} = CCACGC,\ l_{1,3} = CACGCT,$$
$$l_{2,1} = GGACGC,\ l_{2,2} = GACGCT,\ l_{2,2} = GACGCT,\ l_{2,4} = CGCTGC,$$
$$l_{3,1} = GCGCTG,\ l_{3,2} = CGCTGC,\ l_{3,3} = GCTGCA,\ l_{3,4} = CTGCAC,$$

and then compare all these 11 l-mers with each other. The total number of comparisons is up to $\binom{11}{2}$. Noting that we search for unique l-mers which occur approximately

only in one sequence, we only compare l-mers from different sequences and do not compare l-mers occurring in the same sequence. That is, we only compare $l_{i,j}$ and $l_{k,h}$ when $i \neq k$. Thus, the total comparison number is $3 \cdot (4+4) + 4 \cdot 4 = 40$.

First, we compare $l_{1,1}$ with $l_{2,1}$, then with $l_{2,2}$, until we reach the last l-mer, $l_{3,4}$. After all the comparisons for $l_{1,1}$ are performed, we find that no l-mer is exactly and approximately the same as $l_{1,1}$. We say that $l_{1,1}$ occurs only in sequence x_1 and does not occur in x_2 or x_3. By definition, $l_{1,1}$ is unique. The algorithm moves on to the next l-mer, $l_{1,2}$, and compare it with $l_{2,1}$ until $l_{3,4}$. We find that $l_{1,2}$ and $l_{2,1}$ are within 2 mismatches (i.e., $HD(l_{1,2}, l_{2,1}) = 2$). We say that $l_{1,2}$ and $l_{2,1}$ occur approximately at least in two sequences of X, and hence, they are "labeled" to be non-unique. Next, we move to $l_{1,3}$ and compare it with the l-mers in x_2 and x_3, and so on. Note that although $l_{2,1}$ has been marked as non-unique, we still need to compare it with l-mers in x_3, because this helps to decide if l-mers in x_3 are unique. After all the comparisons are carried out, all unique l-mers are picked out and all non-unique l-mers are marked as non-unique. The unique l-mers are $l_{1,1} = ACCACG$, $l_{3,3} = GCTGCA$, and $l_{3,4} = CTGCAC$.

Suppose that we have 5 processors available for parallelization technique. To parallelize the Brute-Force program, we apply the command "ExportEnvironment" to all l-mers such that all processors can work on all l-mers. We write the comparison process for one l-mer into a function and then use the command "ParallelMap". Using the "ParallelMap" command, we send each l-mer to each available processor. A processor takes one l-mer and compares with all the other l-mers to see if it is unique. Then, the processor returns the result to the main computer. The first processor returns that $l_{1,1}$ is unique, the second processor returns that $l_{1,2}$ is not unique, and so

on. Such comparisons can be carried out simultaneously in this manner, thus saving time. As soon as each processor finishes its comparison, it seeks for the next lined up l-mer to perform another comparison until no l-mers are available for comparisons.

For the filtration method algorithm 1, we know that $q = \frac{l}{d+1} = 2$. We have 9 different q-mers in X. All these q-mers and the numbers of their occurrences are listed in Table B.1. The q-mers that occur more than once in X are AC, CA, CG,

	AC	CA	CC	CG	CT	GA	GC	GG	TG
x_1	2	1	1	1	1	0	1	0	0
x_2	1	0	0	1	1	1	2	1	1
x_3	1	1	0	1	1	0	3	0	1

Table B.1: Table for all the 2-mers and their number of occurrences in X (for filtration method 1).

CT, GC, TG. For each of these q-mers, we will find all their locations in X and then extend them to l-mers to determine if those l-mers are non-unique.

Let us take AC for an example. The locations of all AC are $(1,1)$, $(1,4)$, $(2,3)$ and $(3,8)$, where (i,j) refers to position j in x_i. We refer to these q-mers in $(1,1)$, $(1,4)$, $(2,3)$ and $(3,8)$ as $q_{1,1}$, $q_{1,4}$, $q_{2,3}$ and $q_{3,8}$, respectively. According to the definition of unique l-mers, we do not extend $q_{1,1}$ and $q_{1,4}$, because both of them are in the same sequence, x_1. For $q_{1,1}$ and $q_{2,3}$, $q_{1,1}$ starts at the beginning of x_1 and thus, cannot be extended to the left. We can only extend $q_{1,1}$ to the right and obtain one pair of l-mers: $ACCACG$ and $ACGCTG$. $HD(ACCACG, ACGCTG) = 3$ does not

indicate the non-uniqueness of these two l-mers. Note that $q_{3,8}$ ends in x_3, which implies that $q_{1,1}$ cannot be extended to the left and $(3, 8)$ cannot be extended to the right. So, there is no comparison between these two q-mers. For $q_{1,4}$ and $q_{2,3}$, we see that $q_{1,4}$ cannot start or end an l-mer because we extend by q-bases every time. Therefore, there is only one pair of extensions for $q_{1,4}$ and $q_{2,3}$: $CCACGC$ and $GGACGC$. $HD(CCACGC, GGACGC) = 2$, so we record these l-mers $CCACGC$ and $GGACGC$ as non-unique. Next, we take $q_{1,4}$ and $q_{3,8}$ and note that $q_{1,4}$ cannot be extended to the left to obtain an l-mer we do not have a possible extension between these two q-mers. It is the same for $q_{2,3}$ and $q_{3,8}$. From these comparisons, we obtain two non-unique l-mers: $CCACGC$ and $GGACGC$. We carry out the same procedures for other q-mers that occur more than once in the database. This results in the unique oligos of $ACCACG$, $GCTGCA$ and $CTGCAC$.

q-mer	Resulted non-unique l-mers
AC	CCACGC, GGACGC
CA	
CG	CACGCT, CGCTGC, GACGCT
CT	CACGCT, CGCTGC, GACGCT
GC	ACGCTG, CCACGC, CGCTGC, GCGCTG, GGACGC
TG	ACGCTG, GCGCTG

Table B.2: All 2-mers occurring more than once and resulting non-unique l-mers from extensions and comparisons (for filtration method 1).

To parallelize the above filtration method 1, we first find all q-mers that occur more than once (i.e., AC, CA, CG, CT, GC, TG). In our Mathematica program, we write the process of extension and comparison for one q-mer as a function. Using the command "ParallelMap" enable us to send all these q-mers to all different processors for extensions and comparisons. Each processor works on one q-mer at a time simultaneously. Upon completion of extensions and comparisons, each processor returns a list of detected non-unique l-mers. The processor returns nothing if it did not find any non-unique l-mers. The main computer takes all the non-unique l-mers and then finds unique l-mers by eliminating non-unique ones from all the l-mers in the given database.

For the filtration method algorithm 2, $q = \frac{l}{\lfloor \frac{d}{2} \rfloor + 1} = 3$. We search for all q-mers and cluster them into groups such that within each group each q-mer is one mutant to the other q-mers. We have 9 groups for this small example. They are listed in Table B.3. Then we consider q-mers in the same group as the same. This enables us to perform more extensions and comparisons increasing its accuracy to generate unique l-mers.

For the ACC group, the q-mers consist of ACC and ACG. We find the locations for both of them in X as $(1,1)$, $(1,4)$ and $(2,3)$, where (i,j) also corresponds to position j in x_i. We refer to these q-mers in $(1,1)$, $(1,4)$ and $(2,3)$ as $q_{1,1}$, $q_{1,4}$ and $q_{2,3}$, respectively. Since $q_{1,1}$ and $q_{1,4}$ are located in the same sequence, we do not extend them. We extend $q_{1,1}$ and $q_{2,3}$ and obtain one pair of extensions: $ACCACG$ and $ACGCTG$. The Hamming Distance between this pair indicate that they are different ($HD(ACCACG, ACGCTG) = 3$). So we do not record any non-unique l-mer. For $q_{1,4}$ and $q_{2,3}$, we cannot extend them both by q bases. From the first q-mer group, we

q-mer group	Resulted non-unique l-mers
ACC, ACG	
ACG, GCG	ACGCTG, GCGCTG
CAC, CGC	CCACGC, CGCTGC, GGACGC
CAC, GAC	CACGCT, GACGCT
CCA, GCA	
CGC, TGC	CCACGC, CGCTGC, GGACGC
CTG	ACGCTG, GCGCTG
GCT, GCA, GCG	CACGCT, GACGCT
GCA, GGA	

Table B.3: Group table for all 3-mers in X and the resulted non-unique l-mers from extensions and comparisons (for filtration method 2).

do not obtain any non-unique l-mers. From the second group, ACG, we have q-mers consisting of ACG and GCG. These appear in $q_{1,4}$, $q_{2,3}$ and $q_{3,1}$. After each comparison, the Hamming Distance between $q_{2,3}$ and $q_{3,1}$ is 1 (HD(ACGCTG,GCGCTG)=1). Thus, these two l-mers are recorded in the non-unique l-mers list.

We do similar extensions and comparisons for the other q-mer groups and record the resulting non-unique l-mers in Table B.3. This also results in unique l-mers $ACCACG$, $GCTGCA$, and $CTGCAC$. The parallelization for the program for the filtration method 2 is similar to the one for the filtration method 1. Each processor works on one q-mer group for extensions and comparisons.