Patterns of Dipeptide Usage for Gene Prediction

Dayananda Sagar Gangadharaiah
Wright State University

Follow this and additional works at: http://corescholar.libraries.wright.edu/etd_all
Part of the Computer Engineering Commons

Repository Citation
PATTERNS OF DIPEPTIDE USAGE FOR GENE PREDICTION

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Engineering

By

DAYANANDA SAGAR GANGADHARIAH
B.E., Visvesvaraya Technological University, 2005

2010
Wright State University
I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY Dayananda Sagar Gangadharaih ENTITLED Patterns of Dipeptide usage for gene prediction BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF Master of Science in Computer Engineering.

Travis E. Doom, Ph.D.
Thesis Director

Thomas Sudkamp, Ph.D.
Department Chair

Committee on
Final Examination

Travis E. Doom, Ph.D.

Michael L. Raymer, Ph.D.

Sridhar Ramachandran, Ph.D.

John A. Bantle, Ph.D.
Interim Dean, School of Graduate Studies
DEDICATED TO

MOTHER SHARADAMBE- THE GODDESS OF KNOWLEDGE
ABSTRACT

Dayananda Sagar Gangadharaih. M.S.C.E., Department of Computer Science and Engineering, Wright State University, 2010. Patterns of dipeptide usage for gene prediction.

As the number of complete genomes that have been sequenced continues to grow rapidly, the identification of genes regions in DNA sequence data remains one of the most important open problems in bio-informatics. Improving the accuracy of such gene finding tools by a small percentage would affect accurate predictions of many genes of an organism (Zhu et al., 2010). This thesis presents a novel approach for identifying coding regions of a genome based on dipeptide usage.

The patterns in dipeptide usage are used to discriminate between coding and non-coding DNA regions. Two sample T-tests are used as tests of significance to determine the dipeptides that show significant difference in their occurrences in coding and non-coding regions. These methods are primarily tested on Escherichia coli -536 genome, where they reached an accuracy of 96.5% in identifying coding region and 100% accuracy in identifying non-coding regions. The trained classifier data Escherichia coli-536’s genome is utilized to predict the coding and non-coding regions of Salmonella enterica subsp. enterica serovar Typhi’s genome. The results of these experiments showed an accuracy of 79.5% in predicting coding regions and 100% in predicting non-coding regions of Salmonella enterica subsp. enterica serovar Typhi’s genome.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>2. BACKGROUND</td>
<td>7</td>
</tr>
<tr>
<td>2.1. DNA</td>
<td>7</td>
</tr>
<tr>
<td>2.2. Central Dogma of Molecular Biology</td>
<td>9</td>
</tr>
<tr>
<td>2.3. Gene</td>
<td>11</td>
</tr>
<tr>
<td>2.4. Gene Expression and Information Content</td>
<td>13</td>
</tr>
<tr>
<td>2.4.1. Promoter Sequence</td>
<td>13</td>
</tr>
<tr>
<td>2.4.2. The Genetic Code</td>
<td>14</td>
</tr>
<tr>
<td>2.5. Some of Current Methods of Gene Prediction</td>
<td>16</td>
</tr>
<tr>
<td>2.5.1. Content Sensors</td>
<td>16</td>
</tr>
<tr>
<td>2.5.2. Signal Sensors</td>
<td>20</td>
</tr>
<tr>
<td>2.6. Current methods of prokaryotic gene prediction</td>
<td>22</td>
</tr>
<tr>
<td>2.7. T-Test</td>
<td>27</td>
</tr>
<tr>
<td>2.8. Bonferroni’s Correction</td>
<td>27</td>
</tr>
<tr>
<td>2.9. Type 1 and Type 2 Errors</td>
<td>28</td>
</tr>
<tr>
<td>3. IDENTIFICATION OF CODING REGIONS BASED ON NORMALIZED OCCURRENCE VALUES</td>
<td>30</td>
</tr>
<tr>
<td>3.1. Data Collection</td>
<td>31</td>
</tr>
</tbody>
</table>
3.2. Non-coding Region: Separation and Translation........................................ 34
3.2.1. Translating the Non-coding Regions.................................................. 35
3.3. Normalizing the Dipeptide Count.......................................................... 38
3.4. Determining Significance of Difference................................................ 41
3.5. Distinguishing Coding and Non-coding Regions Based on Threshold Method........ 44
3.5.1. Threshold Calculation..................................................................... 44
3.5.2. Coding and Non-coding Region Classification.................................... 45
3.6. Results................................................................................................. 46
3.7. Rejection of Threshold Method for Differentiating Coding and Non-coding...... 48

4. IDENTIFICATION OF CODING REGIONS BASED ON
FREQUENCY DISTRIBUTION........................................................................ 51
4.1.1. Frequency Distribution Patterns....................................................... 51
4.1.2. Selecting the Discriminating Dipeptides.......................................... 56
4.2. Ranking the Translated Genomic Sequence.......................................... 58
4.3. Results................................................................................................. 64

5. CONCLUSION................................................................................................. 66
5.1 Contribution............................................................................................... 66
5.1.1. T-Test.............................................................................................. 67
5.1.2. Detecting the Coding Regions.......................................................... 67
5.2 Comparison of accuracies of various gene predictors.............................. 68
5.3. Future Work............................................................................................ 70
BIBLIOGRAPHY.................................................................................................................. 71

APPENDIX A: C++ PROGRAM TO SEPARATE AND TRANSLATE CODING REGIONS.................................................................................................................. 78

APPENDIX B: C++ PROGRAM TO SEPARATE AND TRANSLATE NON-CODING REGIONS.................................................................................................................. 85

APPENDIX C: C++ PROGRAM FOR T-TEST........................................................................ 89

APPENDIX D: C++ PROGRAM FOR GENERATING FREQUENCY DISTRIBUTION OF SIGNIFICANT DIPEPTIDES................................................................. 96

APPENDIX E: C++ PROGRAM FOR RANKING THE GENOMIC STRINGS........ 101

APPENDIX F: LIST OF SIGNIFICANT DIPEPTIDES RANKED AS PER RESPECTIVE CUMULATIVE CODING WEIGHTS................................................................. 105
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1.</td>
<td>DNA Structure</td>
<td>8</td>
</tr>
<tr>
<td>2.2.</td>
<td>The Central Dogma of Molecular Biology</td>
<td>10</td>
</tr>
<tr>
<td>2.3.</td>
<td>Gene Structure</td>
<td>12</td>
</tr>
<tr>
<td>2.4.</td>
<td>Amino Acid Chart</td>
<td>15</td>
</tr>
<tr>
<td>3.1.</td>
<td>Contents of NCBI genome URL</td>
<td>32</td>
</tr>
<tr>
<td>3.2.</td>
<td>Reading the genes</td>
<td>33</td>
</tr>
<tr>
<td>3.3.</td>
<td>Translating the non-coding regions</td>
<td>37</td>
</tr>
<tr>
<td>3.4.</td>
<td>Detecting dipeptides</td>
<td>38</td>
</tr>
<tr>
<td>3.5.</td>
<td>Normalizing the dipeptide counts</td>
<td>40</td>
</tr>
<tr>
<td>3.6.</td>
<td>Example of Normalized Occurrence table</td>
<td>40</td>
</tr>
</tbody>
</table>
3.7. Significant Dipeptides................................................................. 42
3.8. Threshold Calculation............................................................... 45
3.9. Calculating error rate................................................................. 47
3.10. Unacceptable Type 1 and Type 2 errors.................................... 49
4.1. Generating Frequency Distribution............................................. 54
4.2. Ranking the genomic strings...................................................... 61
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1. Dipeptide and respective Type1 and Type2 errors</td>
<td>48</td>
</tr>
<tr>
<td>4.1. Frequency distribution table</td>
<td>55</td>
</tr>
<tr>
<td>4.2. Example of potential coding identifiers</td>
<td>56</td>
</tr>
<tr>
<td>4.3. Example of non-potential coding identifiers</td>
<td>59</td>
</tr>
<tr>
<td>4.4. Ranking of genomic strings</td>
<td>62</td>
</tr>
<tr>
<td>4.5. Type1 and Type 2 errors in <em>E.coli</em></td>
<td>65</td>
</tr>
<tr>
<td>4.6. Type 1 and Type 2 errors in Salmonella</td>
<td>65</td>
</tr>
<tr>
<td>5.1 Accuracy of gene prediction</td>
<td>69</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

The decoding of the human genome paved the way for a new era of biotechnology. Understanding and interpreting very large genome data sets, predicting the structures and functions of proteins and designing drug molecules for target proteins constitute some of the challenges in science. Bioinformatics brings computer scientists, mathematicians, computer engineers to work together with biologists to tackle such problems.

All organisms self replicate due to the presence of genetic material called DNA. The entire DNA content of the cell is known as the genome. The segment of the genome from which the proteins are ultimately made is called the gene (Shenoy et al., 2006). Understanding these genes is one of the modern day challenges. Why only a small percentage of the entire DNA forms the genes and what is the rest of the DNA responsible for, under what conditions genes are expressed, where, when, and how to regulate gene expressions, are some unsolved puzzles.

Genome data that is becoming available at an accelerated pace poses challenges to computer scientist in dealing with data storage, data mining, and other database management issues.
Bioinformatics involves discovery, development, and implementation of algorithms and software tools that facilitate an understanding of biological processes.

Bioinformatics has a key role to play in areas like agriculture where it can be used for increasing the nutritional content, increasing the volume of agricultural products, implementing disease resistance, etc (Shenoy et al., 2006). In the pharmaceuticals sector, it can be used to reduce the time and cost involved in drug discovery process and to develop personalized medicine (Shenoy et al., 2006). One of the primary challenges for these applications lies in identifying genes; which carry the information for synthesizing proteins.

Gene recognition involves identification of stretches of sequence, usually DNA, that are biologically functional. This not only includes the protein coding genes, but also other functional elements such as RNA genes and regulatory regions. Gene recognition is the most important step in understanding the genome of a species once it has been sequenced.

The existence of genes was first suggested by Gregor Mendel (1822-1884) based on his study of inheritance in pea plant. In 1972, Walter Fiers and his team at the Laboratory of Molecular Biology of the University of Ghent (Ghent, Belgium) were the first to determine the sequence of a gene: the gene for Bacteriophage MS2 coat protein (Jou et al., 1972). In its earliest days, gene recognition was based on experimentation on living cells and organisms. Statistical analysis was made to determine the order of several genes on a certain chromosome, and information from many such experiments was combined to create a genetic map specifying the rough location of genes relative to each other. Today, with the advancement of technology and new computational
recourses at the disposal of the researchers, gene recognition has been redefined as a large computational problem.

Gene recognition methods are broadly classified into two categories: The extrinsic approach and the Ab Initio approach (Bandyopadhyay et al., 2008). In extrinsic (or evidence-based) approach, the target genome is searched for sequences that are similar to known sequence of a messenger RNA (mRNA) or protein product. BLAST is a widely used system designed for this purpose. In the Ab Initio approach, genomic DNA sequence is searched for certain signs of protein-coding genes. These characteristic signs could be either signal, specific sequences that indicate the presence of a gene nearby, or content, statistical properties of protein-coding sequence itself. In prokaryotic genome, genes have specific and relatively well understood promoter sequences (signals) that mark transcription start sites. In the eukaryotes, the classic signals for gene regions are the GC islands (regions of high content of G and C) and the poly (A) tail (contiguous stretches of A’s).

The contemporary gene recognition tools make use of complex probabilistic models such as Markov chains and Hidden Markov Models (Borodovsky, 1998, Yada et al., 1999). Using a Markov chain, one can calculate the probability that a given sequence of DNA of a prokaryotic genome is a coding region. More specifically it helps in calculating transition probabilities: the probability of an amino acid being followed by another amino acid.

Our thesis is focused on investigating the effectiveness of dipeptide usage for differentiating coding and non-coding regions of a genome. Our approach is based on a simple idea of
determining the dipeptides that show significant difference in their occurrences in coding and non-coding regions. Based on the frequency distribution of occurrences of these dipeptides, we will determine the threshold of number of dipeptide identifiers for discriminating coding regions from the rest of the genome.

Our approach is validated in collecting the Escherichia_coli_536 genome data from the NCBI website and calculating the normalized occurrence of the dipeptides in the coding and non-coding regions. Two sample T-tests are used as the test of significance to determine the dipeptide with significant difference of occurrences between coding and non-coding regions. Considering the dipeptides with significant difference in their occurrences, we determine the frequency distribution of these dipeptides for randomly selected segments from coding and non-coding regions. Based on the frequency distribution, we determine the threshold of the number of dipeptide identifiers for discriminating coding and non-coding regions. Having studied the performance of our algorithm on the E.coli’s genome, we test our algorithm on the Salmonella’s genome (which is a genetically closer relative of E.coli) to determine if coding and non-coding regions of Salmonella’s genome can be discriminated using the frequency distribution data and the threshold of E.coli’s genome. We also find the accuracy of identifying the coding and non-coding regions of Salmonella’s genome.

The remainder of this thesis is organized as follows: Chapter 2 details the background material for the ensuing chapters. Chapter 3 describes the methods for calculating the normalized values of the dipeptide occurrences, the T-tests and a naïve classification using this information.
Chapter 4 presents the methods for determining frequency distribution patterns of the significant dipeptides. Chapter 4 further describes the methods for selecting and ranking the coding dipeptide identifiers and determining the threshold of number of dipeptide identifiers for identifying coding and the non-coding regions based on the frequency distribution of the significant dipeptides. This threshold is used for calculating the Type 1 and Type 2 errors in identifying randomly selected coding and non-coding regions of *E.coli*. The results generated for *E.coli’s* genome are validated by testing the performance of our algorithm on Salmonella’s genome using the frequency distribution data and threshold of *E.coli’s* genome. Chapter 5 concludes the thesis with the contributions of this research and the scope for possible future work on the basis of the findings of the research.
Chapter 2

Background

This chapter describes the necessary concepts required for understanding remaining chapters. Section 2.1 explains the structure and importance of DNA. Section 2.2 describes the Central dogma of molecular biology. Section 2.3 describes the prokaryotic and eukaryotic genes. Section 2.4 deals with genes structure and information context. Section 2.5 describes some of the current gene prediction techniques. In section 2.6, T-test and Bonferroni’s correction on a high level is described. Finally in Section 2.7, Type 1 and Type 2 errors are described.

2.1 DNA

Deoxyribonucleic acid (DNA) is a nucleic acid that contains the genetic instructions which enable all organisms to replicate. Chemically, DNA consists of two long sequences of simple unit called nucleotides. The two nucleotide strands entwine themselves in the shape of a double helix, that is, each is shaped like a spiral staircase. The two strands bound together make DNA a double helix.

A nucleotide is made up of a base and a sugar linked to it. There are four different bases constituting the DNA; Adenine (A), Thymine (T), Guanine (G), and Cytosine (C). These bases
attach to sugar/phosphate to form the complete nucleotide. A base on a DNA strand interacts with a base on the other DNA strand. These bases are held by hydrogen bonds. The nucleotide bases are classified into two types: purines and pyrimidines. Adenine and Guanine (fused five- and six-membered rings) are called purines, and cytosine and thymine (six-membered rings) are the pyrimidines. Every G forms three hydrogen bonds with C on the complementary DNA strand and vice versa. Similarly, every A forms two hydrogen bonds with the T on the complementary DNA strand and vice versa. The other combinations of the bases do not usually take place due to their chemical incompatibility. The structure of DNA is shown in Figure 2.1 (Encyclopedia Britannica, 1998).
In a double helix, the direction of the nucleotides on one strand is opposite to their direction on the complementary strand and therefore said to be anti parallel to one another. The ends of DNA strands are referred to as 5’ and 3’ ends. The 5’ end terminates with a phosphate group and the 3’ end with a hydroxyl group.

Within cells, DNA is organized in the chromosomes. The chromosomes get duplicated before cells divide, in a process called DNA replication. In eukaryotes (animals, plants, fungi, and protists) the DNA resides in the cell nucleus, while in prokaryotes (bacteria and archae) it is found in the cell's cytoplasm. Within the chromosomes, proteins called histones organize DNA. These compact structures guide the interactions between DNA and other proteins, helping control which parts of the DNA are to be transcribed.

2.2 Central Dogma of Molecular Biology

The process by which information is extracted from DNA to make a protein is called Central Dogma of Molecular Biology. Figure 2.2 (source: www.physics.arizona.edu/~skinner) describes central dogma. The information in the DNA is used to make a transient, single stranded, polynucleotide called RNA. This process is called transcription. Transcription takes place in 5’ to 3’ direction. There is a one-to-one correspondence between the bases used to make
Figure 2.2: The Central Dogma of Molecular Biology - The process of protein synthesis

RNA and the bases in the nucleotide sequence of DNA except that in RNA the base uracil (U) takes place of thymine (T). This urasil differs from thymine by lacking a methyl group on its ring. The process of transcription is accomplished through the enzymatic activity of RNA polymerase II.
The process of converting the information from nucleotide sequence in RNA to the amino acid sequence that make a protein is called translation. This process is performed by a complex protein called ribosome and the t- RNA.

2.3 Gene

Genes are regions of DNA that encode for proteins. In cells, a gene is a portion of an organism's DNA which contains both "coding" sequences that determine what the gene does, and regulatory sequences that determine when the gene is active (expressed), and “non-coding” (junk) sequences.

All genes have regulatory regions called promoters that mark the start of transcription. A promoter provides the position that is recognized by the transcription machinery when a gene is about to be transcribed and expressed. A gene can have more than one promoter, resulting in RNAs with varying lengths. The promoter sequences of eukaryotes are more complex than the prokaryotes. The small segments before and after the coding regions are called the Un-translated regions (UTR) which get transcribed but not translated. In the prokaryotes the translation begins when a ribosome encounters the start codon and ends when the ribosome has reached the stop codon. In the eukaryotes this process is a bit complex; the transcribed m-RNA has long stretches of base pairs called introns that never get translated into protein. The actual sections of the m-
RNA that get translated are called exons. Prior to translation a process called splicing takes place, which involves precise excision of the introns and rejoining of the exons. The structures of eukaryotic and prokaryotic genes are shown in Figure 2.3.

---

**Eukaryotic DNA**

![Eukaryotic DNA Diagram](image)

**Prokaryotic DNA**

![Prokaryotic DNA Diagram](image)

Figure 2.3: Gene Structure- The structure of prokaryotic and eukaryotic DNA
2.4 Gene Expression and Information Content

This section discusses the process of gene expression and the process of encoding the codons to different amino acids. The sub-section 2.4.1 discusses about the promoter sequence and the sub-section 2.4.2 discusses the process of translating the information contained in the mRNA into proteins.

2.4.1 Promoter Sequence

Gene expression involves processing the information in DNA to transcribe to RNA and then translate to corresponding protein. There are two factors that cells of an organism must emphasize while controlling the gene expression. First, they must be able to distinguish the part of the genome corresponding to the gene. Second, cells must be able to determine which gene is to be expressed at a given time.

Since the gene expression is initiated by the RNA polymerase II, the RNA polymerase II is responsible for making these two distinctions. The RNA polymerase II scans the DNA looking for a specific sequence of the nucleotides which mark the beginning of the gene. This sequence is called the promoter sequence. The DNA transcription ends when the RNA polymerase II encounters a specific sequence of nucleotides. This region is called the transcription stop site.
The expression of a gene is regulated by specific proteins called the regulatory proteins. These proteins bind to a specific sequence of nucleotides depending on the need for a particular gene expression. When binding of regulatory protein initiates transcription by the RNA polymerase II, a positive regulation is said to have occurred. Binding of a regulatory protein inhibiting transcription, results in negative regulation.

### 2.4.2 The Genetic code

The transcribed RNA is translated by the ribosome, into a chain of amino acids which are the building blocks of proteins. The function of protein is dependent on the order in which its amino acids are linked. The ribosomes take the responsibility of translating the four nucleotide code of the transcribed RNA to a long sequence of a 20 different amino acids. It is obvious that, there is no one to one correspondence of single nucleotide and the amino acids that are to be encoded. Considering two nucleotide sequences result in 16 unique sequences; still insufficient for encoding the 20 amino acids. Ribosomes consider triplet of nucleotides to translate the information in RNA to amino acid sequence. These triplets of nucleotides are called the codons. The triplet of nucleotides gives rise to $4^3 = 64$ codons. Figure 2.4 shows that there is more than one codon coding for a given amino acid.
However three of the codons (UAA, UAG, UGA) do not code for any amino acid but they are responsible in termination the translation process. These codons are called Stop codons. The translation is initiated when the ribosomes encounter a specific codon called the start codon. This start codon is AUG and it also codes for amino acid Methionine.
2.5 Some of Current methods of gene predictions

Gene prediction refers to the area of computational biology concerned with locating stretches of genomics DNA that are biologically functional. Gene prediction is one of the foremost basic steps in understanding the genome of a species which has been sequenced.

There are two different types of information currently used to locate gene in the genomic sequence namely, content sensors and signals sensors. Content sensors are measures that try to classify a DNA region into coding and non-coding regions. Historically, the existence of a sufficient similarity with an already characterized sequence has been the means of obtaining such a classification. Signal sensors are measures that try to detect the functional sites specific to a gene (Mathe et al., 2002).

2.5.1 Content Sensors

The content sensors are characterized as extrinsic and intrinsic content sensors.

The extrinsic content sensors simply compare a given genome sequence region and a protein or DNA sequence in the database in order to determine whether the region in question is a coding region. This similarity can be detected using the local alignment tool such as or the optimal Smith-
Waterman algorithm, fast heuristic approaches such as FASTA (Pearson and Lipman, 1988) and BLAST (Altschul et al., 1990). The demerit the extrinsic approach is that nothing can be found if the database does not contain a sufficiently similar sequence. The important aspect of similarity based predictions depends on the previously accumulated biological data. They should thus produce biologically relevant predictions. Another important point is that a single match is enough to detect the presence of a gene, even if it is not canonical. The extrinsic content sensors are more used on eukaryotic genomes (Mathe et al., 2002).

The other category of content sensors is the intrinsic content sensors. Originally, intrinsic content sensors were defined for prokaryotic genomes. In such genomes, only two regions are considered: the regions that code for a protein and will be translated and intergenic regions (the regions between two transcribed regions). Since the coding regions will be transcribed, they will be characterized by considering codons, which will be translated into a specific amino acid in the final protein.

In prokaryotic sequences, genes define (long) uninterrupted coding regions that must not contain stop codons. Thus the straight forward approach for finding coding regions is to look for long open reading frames (ORF’s), defined as sequences not containing stops, i.e. as sequences, between a start and stop codon. In eukaryotic sequences, the translated regions may be very short, and the absence of stop codons becomes meaningless (Fickett, 1995).

Several other measure have therefore been defined that try to more finely characterize the fact that a sequences is coding for a protein: nucleotide composition and especially (G+C) content
(introns being more A/T rich than exons, especially in plants), codon composition, hexamer frequency, base occurrence periodicity, etc (Mathe et al., 2002). Among the large variety of coding measures that have been tested, hexamer frequency (i.e. usage of 6 nucleotide long words) was shown in 1992 to be the most discriminative variable between coding and non-coding and sequences (Fickett and Tung, 1992). This characteristic has been widely exploited by a large number of algorithms through different methods (Mathe et al., 2002).

Thus, hexamer frequency is one of the main variables used in SORFIND (Hutchinson and Hayden, 1992), Geneview2 (Milanesi et al., 1993), the quadratic discriminant approach of MZEF (Zhang, 1997) and the neural network procedure of GeneParser (Snyder and Stormo, 1995). This last program combines the use of hexamer frequency with local compositional complexity measures estimated on octanucleotide statistics. Such statistics are also efficiently used, among other variables, in the linear discriminate analysis of Solovyev’s Gene Finder (Solovyev and Salamov, 1997).

More generally, the kmer composition of coding sequences is the basis of ‘three periodic Markov model’ used in the Genemark algorithm (Borodosky and McIninch, 1993). A Markov model is a model which assumes that the probability of appearance of a given base (A, T, G or C) at a given position depends only on the k previous nucleotides (k is called the order of the Markov model). Such a model is defined by the conditional probabilities \( P(X/k \text{ previous nucleotides}) \), where \( X = A, T, G \text{ or } C \). In order to build a Markov model, a learning set of sequences on which these probabilities will be estimated is required. Given a sequence and a Markov model, one can then
very simply compute the probability that this sequence has been generated according to this
model, i.e. the likelihood of the sequence, given the model (Mathe et al., 2002).

The simplest Markov models are homogeneous zero order Markov models which assume that
each base occurs independently with a given frequency. Such models are used for non-coding
regions, however the more recent algorithms like Genemark, Genscan (Burge and Karlin, 1997)
and EuGene (Sehiex et al., 2001) use higher order models to represent introns and intergenic
regions. The more complex three periodic Markov models have been introduced to characterize
coding sequences. Coding regions are defined by three Markov models, one for each position
inside a codon.

The larger the Markov model, the finer it can characterize dependencies between adjacent
nucleotides. However, a model of order k requires a very large number of coding sequences to
reliably estimate. Therefore, most gene prediction programs, such as Genemark and Genscan,
usually rely on a three-periodic Markov model of order five (thus exploiting hexamer
composition) or less to characterize coding sequences (Mathe et al., 2002). To cope with these
limitations, interpolated Markov models (IMMs) have been introduced in the prokaryotic gene
finder Glimmer (Salzberg et al., 1998). For each conditional probability, an IMM combines
statistics from several Markov models, from order zero to a given order k (typically k=8),
according to the information available. These IMMs are also used in Glimmer (Salzberg et al.,
1999), a version dedicated to eukaryotes, and in EuGene. The later version of Glimmer
introduces yet another sophistication of Markov models called interpolated context models, which can capture dependencies among 12 adjacent nucleotides (Delcher et al., 1999).

Another type of refinement is often needed in eukaryotic genomes. It consists of estimating several gene models according to the G+C content of the genomic sequence. This is done by Genescan and GeneMark.hmm (Lukashin and Borodovsky, 1998).

Although these methods are considered as ‘intrinsic’, the fact that the models are built from known sequences will inherently limit the applicability of the methods to sequences that, globally, behave in the same way as the learning set (Mathe et al., 2002).

### 2.5.2 Signal sensors

Searching for a match with a consensus sequence would be the basic approach of finding a signal that may represent the presence of a functional site, the consensus being determined from a multiple alignment of functionally related documented sequences (Mathe et al., 2002). This type of method is used for splice sites prediction in SPLICEVIEW (Rogozin and Milannese, 1997) and Splice Predictor (Kleffe et al., 1996).

A better representation of signals is presented by the positional weight matrices (PWMs), which indicate the probability that a given base appears at each position of the signal (again computed
from a multiple alignment of functionally related sequences). Equivalently, one can say that a PWM is defined by one classical zero order Markov model per position, which is called an inhomogeneous zero order Markov model (Mathe et al., 2002). The PWM weights can also be optimized by a neural network method, as proposed by Brunak et al (Brunak et al., 1991) for NetPlantGene (Hebsgaard et al., 1996) and NetGene2 (Tolstrup et al., 1997) and used in NNSplice (Reese et al., 1997).

In order to capture possible dependencies between adjacent positions of a signal, one may use higher order Markov models weight array model (WAM). It was first proposed by Zhang and Marr (Zhang, M. Q. and Marr, T.G., 1993) and later used by Salzberg (Salzberg, 1997) it in VEIL (Henderson et al., 1997) and MORGAN (Salzberg et al., 1998) software. Genscan also uses a modified WAM to represent branch point information. This is closely related to position-dependent triplet frequency model employed by MZEF for the some signal (Mathe et al., 2002).

A similar approach is used in GeneSplicer, which combines MDD models for splice sites with second order Markov models that characterize coding / non-coding regions around splice sites. It has been shown that combining sequence based metrics for splice sites (WAM) with secondary structure metrics could lead to valuable improvements in splice site prediction (Patterson et al., 2002).

The main purpose of such programs is not to find the gene structure but to try to find the correct exon boundaries. They are thus very useful in addition to an exon or gene predictor in order to refine an existing gene structure. These programs can also provide insights into possible
alternative splicing, even if, so far, this possibility has been very poorly investigated (Mathe et al., 2002).

Finally, HMMs have also been used to represent other types of signals, such as poly (A) sites (in 3'-UTRs), promoters, etc. As for the intrinsic content sensors, the fact that HMMs are built from a multiple alignment of known functional sequences inherently limits the sensors to canonical signals (Mathe et al., 2002).

Another important signal to identify when trying to predict a coding sequence is the translation initiation codon. A few programs exist specifically dedicated to this problem (Zien et al., 2000) (Nishikawa et al., 2000), but most of them have a rather limited efficiency, which is maybe related to the lack of proper learning sets for eukaryotic genomes. Experimental information on the genuine location of translation starts has indeed been scarce up to now, a situation that will likely change soon with the advent of proteome data (Mathe et al., 2002).

2.6 Current methods of prokaryotic gene prediction

Computational gene finders can be divided into two classes: intrinsic and extrinsic. Intrinsic, or ab initio, gene finders make no explicit use of information about DNAs or proteins outside the sequence being studied. Extrinsic gene finders utilize sequence similarity search methods to
identify the locations of protein-coding regions. In many gene prediction projects, both of these methods are used in conjunction.

In April 2010, Zhu et al., published an ab initio gene detection algorithm in metagenomic sequences. Accurate ab initio gene prediction in a short nucleotide sequence of anonymous origin is hampered by uncertainty in model parameters. One of the effective machine learning methods to bypass this problem is to estimate parameters from dependencies, formed in evolution, between frequencies of oligonucleotides in protein-coding regions and genome nucleotide composition. With advent of new prokaryotic genomes en masse it became possible to enhance this approach by using direct polynomial and logistic approximations of oligonucleotide frequencies, as well as by separating models for bacteria and archaea. These advances have increased the accuracy of model reconstruction and, subsequently, gene prediction.
The standard tools for \textit{ab initio} prokaryotic gene prediction like EasyGene (Larsen and Krogh, 2003), GeneMarkS (Besemer et al., 2001) or Glimmer (Delcher et al., 2007) were not designed to work with short sequence fragments from unknown genomes. However, Zhu et al., used a special ‘heuristic model’ method for assignment of parameters for accurate gene finding in short prokaryotic sequences. The important observation made upon analysis of 17 genomes (Besemer and Borodovsky., 1999) was that frequencies of nucleotides in the three codon positions depend linearly (though with distinctly different slope coefficients) on global nucleotide frequencies. This observation means that nucleotide frequencies in the three codon positions depend linearly on genomic GC content. These linear functions were used to reconstruct codon frequencies in the whole genome using information derived from its short sequence fragment and to derive parameters of the ‘heuristic’ second-order Markov models [the Heuristic Algorithm (HAL)-99 models] for a gene finding algorithm. The work of Zhu et al., assessed the accuracy of a hidden Markov model (HMM) based gene finder, GeneMark.hmm, using the new models on the sets of short sequences obtained by splitting known genomes into equal length fragments. This assessment showed a higher accuracy in comparison with several other existing gene finding methods.

In 2006, Noguchi et al., published their algorithm-MetaGene, for prokaryotic gene finding. MetaGene utilizes di-codon frequencies estimated by the GC content of a given sequence with other various measures. MetaGene can predict a whole range of prokaryotic genes based on the anonymous genomic sequences of a few hundred bases. For non supervised gene finding in
fragmented anonymous sequences, a heuristic model is to be used (Besemer and Borodovsky, 1999). In this model, codon frequencies are approximated by the GC content of a given genome. MetaGene extends this method to estimate dicodon frequencies and achieve higher prediction accuracy than results using mono codon frequencies. In addition to dicodon frequencies, methods such as frequency distribution of ORF lengths, the distance from leftmost start codons, and the distances between neighboring ORFs, are incorporated in MetaGene. These additional methods result in a sensitivity of 95% and a specificity of 90% for shotgun sequences (700 bp fragments from 12 species).

MetaGene predicts gene in two stages. In the first stage, the possible ORFs are extracted from a given sequence and are scored according to their base compositions and lengths. The authors of MetaGene define an ORF as a sequence of codons starting from a start codon and stopping at a stop codon. In the second stage an optimal combination of ORFs is calculated using the scores of orientations (depends on whether ORF is on original strand or complimentary strand) and distances of neighboring ORFs in addition to the scores for the ORFs themselves. This two-stage approach also allows us to predict overlapping genes with appropriate scores.

The gene recognition tool CRITICA (Coding Region Identification Tool Invoking Comparative Analysis), also uses the dicodons for identifying the likely protein-coding sequence (Badger and Olsen, 1999). CRITICA is a suite of programs formed by combining comparative analysis of DNA sequences with non comparative methods. For comparative analysis, DNA sequences are aligned with related sequences from DNA databases; a coding region is interpreted if the
translation of aligned sequences has greater amino acid identity than expected for the observed percentage nucleotide identity. For non comparative analysis, information is derived from the relative frequencies of dicodons in coding frames versus other contexts by iterative analysis of the data (Badger and Olsen, 1999).

CRITICA analyses a given DNA sequence in four steps. In the first stage, each trinucleotide (triplet) in the query DNA is assigned a numerical score based on how much more it resembles a codon in a coding sequence than it resembles a triplet in a noncoding region. This score is a sum of two components: a comparative score based on the relative identities of the nucleotides and the corresponding potential amino acids and a noncomparative score based on dicodon bias in coding frames. In second stage, the tool identifies regions of sequence that have higher-than random scores for coding. In the third stage, the candidate coding region is extended to a terminator codon or to the end of the query sequence. Finally, the effect of choosing each of the available initiator codons is examined by incorporating an initiator codon preference score and a score for any potential ribosome-binding site. If the resulting overall evidence of coding is sufficiently high, the DNA sequence is predicted to be coding.

The results of the above methods shall be compared with that of our algorithm in the conclusion chapter.
2.7 T-Test

T-test assesses whether the difference in the means of two populations is significant or not. We use t-test to determine if difference in the normalized values of a given dipeptide in coding and non-coding region is significant or not.

T-test is mainly used to test the null hypothesis $h_0$ which is also called the hypothesis of no difference. For testing the hypothesis $h_0$ that there is no significant difference between the two populations; we have to show the $t_{\text{calculated}} \leq t_{\text{tabled}}$ (Gupta and Kapoor, 1980). This test will be conducted with some degree of freedom (usually 95%), which considers the errors due to random noise.

2.8 Bonferroni Correction

When several dependent or independent statistical tests are performed simultaneously, a given alpha value (the degree of freedom) of an individual comparison may not be appropriate for the set of all comparisons. In order to reduce the error due to multiple comparisons the alpha value has to be lowered depending on the number of comparisons being performed.
Bonferroni’s correction is one such multiple comparison corrections that set the alpha value for the entire set of n comparisons equal to alpha by taking the alpha value for each comparison equal to $\alpha/n$ (Glantz, 2001).

### 2.9 Type 1 and 2 Errors:

The type 1 and 2 errors are also called as false positive ($\alpha$- error) and false negative ($\beta$- error). These terms are used to describe the errors made in statistical decision process.

The type 1 error or false positive occurs due to rejecting a null hypothesis when the null hypothesis is actually true. An example of this would be; a test for detecting coding region where in the algorithm wrongly predicts a non-coding region to coding region.

The type 2 error or false negative occurs due to considering a null hypothesis to be true (i.e. test failing to reject the null hypothesis) when it is actually false. An example of this would be if the algorithm wrongly predicts a coding region to be non-coding when the null hypothesis is to check for coding region.
These two errors depend on the manner in which the null hypothesis is considered. For example, type 1 & 2 errors change depending on whether the test is for detecting coding region or the non-coding region.

Related calculations to type 1 & 2 errors are specificity and sensitivity. Sensitivity refers to the actual positives that are correctly identified as such, whereas, the specificity refers to actual negatives that are correctly identified by an experimental process.
Chapter 3

Identification of coding regions based on normalized occurrence values

Coding regions of a genome differs from non-coding region in many respects. Identifying this difference is crucial for gene recognition.

In this chapter we describe the methods incorporated in our thesis for distinguishing coding region from the non-coding region. Section 3.1 describes the steps for collecting the Escherichia_coli_536 genome data from the NCBI website. Section 3.2 details the methods for identifying and translating the non-coding region. Section 3.3 presents the algorithm for normalizing the dipeptide count. Section 3.4 describes the t-test, which is used to determine the dipeptides with significant difference of occurrences in coding and non-coding regions. Section 3.5 describes the calculation of type 1 and type 2 errors for predicting the coding and non-coding regions based on the threshold value of a given dipeptide occurrence and in the end; section 3.6 details the reasons for rejecting the method of predicting coding and non-coding regions based on the threshold values of a dipeptide occurrence.
3.1 Data collection

Our initial research focuses on analyzing and testing our algorithms on the Escherichia_coli_536 genome data from the NCBI website. The choice on *E.coli* was made based on the fact that *E.coli* are the simplest prokaryotes and a well studied genome.


The file with ‘.gbk’ extension provides important information about the genome like the organism name, source, locus of coding regions and genes, gene names, translated segments of genes and the proteins formed from these genes. Figure 3.1 shows the starting few lines of the ‘.gbk’ file of the *E.coli*-536.

The actual sequence of the translated gene is presented between a pair of double quotes (‘ ’). Figure 3.1 shows one of the translated gene sequences. Our program reads the ‘.gbk’ file until it encounters the word ‘/translation=’ and there after it stores the translated gene sequence in an array. Every time when a new translated gene is detected it is added to the end of the array. Thus, at the end of this process, the array will contain an ordered sequence of all the translated genes placed one behind the other. This array is used to calculate the normalized dipeptide counts,
which will be discussed in Section 3.3. Figure 3.2 shows the algorithm used for joining all the translated gene sequences into one array and then saving the array in a file.

```
LOCUS CP000247 4938920 bp DNA circular BCT 13-JUL-2006
DEFINITION Escherichia coli 536, complete genome.
ACCESSION CP000247
VERSION CP000247.1 GI:110341805
KEYWORDS .
SOURCE Escherichia coli 536
ORGANISM Escherichia coli 536
  Bacteria; Proteobacteria; Gammaproteobacteria; Enterobacteriales; Enterobacteriaceae; Escherichia.
REFERENCE 1 (bases 1 to 4938920)
  TITLE Role of pathogenicity island-associated integrases in the genome plasticity of uropathogenic Escherichia coli strain 536
REFERENCE 2 (bases 1 to 4938920)
  TITLE Direct Submission
  JOURNAL Submitted (20-JAN-2006) Institute of Microbiology and Genetics, Georg August University Goettingen, Goettingen Genomics Laboratory, Grisebachstr. 8, Goettingen D-37077, Germany
FEATURES Location/Qualifiers
  source 1..4938920
    /organism="Escherichia coli 536"
    /mol_type="genomic DNA"
    /strain="536"
    /db_xref="taxon:362663"
    /note="serogroup: O6:K15:H31"
  gene 190..255
    /locus_tag="ECP_0001"
    /product="thr operon leader peptide"
    /protein_id="ABG68043.1"
    /translation="MKRISTTITTTITITTGNGAG"
```

Figure 3.1: Contents of NCBI genome URL- Structure of ‘.gbk’ file of the *E.coli* genome
The file with ‘.fna’ extension consists of the entire genome sequence. This file can be used for analyzing genome. In our work we use the ‘.fna’ file for separating the non-coding regions from the genome. The following section describes our approach for separating the non-coding regions from the genome and translating these sequences.

Figure 3.2: Reading the genes-Flow chart for separating the translated gene sequences from ‘.gbk’ file
3.2 Non-coding region: Separation and Translation:

*E.coli*, being prokaryotic organisms has a low density of non-coding regions. In our work we analyze the coding and non-coding regions separately. Thus, we had to separate the non-coding region from the rest of the genome. The starting and ending locations of each coding region is determined from the ‘.gbk’ file and from the ‘.fna’ file, the nucleotides present in these locations are replaced by a “+” sign. The rest of the genome is considered as the non-coding regions. This modified ‘.fna’ file is saved as a new file which is used for translation of the non-coding region. Replacing the coding nucleotides by “+” sign instead of completely eliminating them, ensures that all non-coding nucleotides retain their positional values.

The non-coding region can be translated into six different reading frames (three reading frames in 5'-3' direction and three reading frames in 3'-5' direction). A complimentary non-coding region sequence is created to obtain three reading frames in the reverse direction (i.e. in the 3’-5’ direction). For generating the complimentary sequence, the entire modified ‘.fna’ file is first copied into a string array in the reverse order and then the nucleotides A,G,U, C are replaced by U, C,A,G respectively.
3.2.1 Translating the Non-coding regions:

The reading frames are generated by grouping triplets of nucleotides in different order. Translation to the first reading frame is done by grouping triplets of nucleotides starting from the first nucleotide in the non-coding sequence. Each of these triplets is then coded to the corresponding amino acid. The second and the third reading frames are generated by considering the translation start site at the 2nd and 3rd nucleotide respectively. The same algorithm is used on the complementary non-coding sequence to generate the other three reading frames. When the triplet grouping window encounters a “+” sign, it is treated as the beginning of the coding region and hence pauses translating until the window passes over all the “+” signs and returns back to a region with only (i.e., translation ceases until there is no “+” sign in the grouping window). Figure 3.3 shows the flow chart for generating all the six reading frames.

The 6 reading frames generated are saved in 6 individual files. These files serve as the data files for the other sections of our research.
Start the index $i$ from the 1st nucleotide on the non-coding sequence

Read $(i+1)$th and $(i+2)$th nucleotides

Check if at least one “+” sign is present in the triplet

Yes

No

Translate the nucleotide in the $i$, $(i+1)$th and $(i+2)$th positions to the corresponding amino acid

Move the translation window by 3 nucleotides and assign the start position of the new window to $i$

End of sequence?

Yes

End of translation

No

Start the index $i$ from the 2nd nucleotide on the non-coding sequence

Read $(i+1)$th and $(i+2)$th nucleotides

Check if at least one “+” sign is present in the triplet

Yes

No

Translate the nucleotide in the $i$, $(i+1)$th and $(i+2)$th positions to the corresponding amino acid

Move the translation window by 3 nucleotides and assign the start position of the new window to $i$

End of sequence?

Yes

End of translation

No

Start the index $i$ from the 3rd nucleotide on the non-coding sequence

Read $(i+1)$th and $(i+2)$th nucleotides

Check if at least one “+” sign is present in the triplet

Yes

No

Translate the nucleotide in the $i$, $(i+1)$th and $(i+2)$th positions to the corresponding amino acid

Move the translation window by 3 nucleotides and assign the start position of the new window to $i$

End of sequence?

Yes

End of translation

No

Figure 3.3 (a): Flow chart for generating reading frame 1

Figure 3.3 (b): Flow chart for generating reading frame 2

Figure 3.3 (c): Flow chart for generating reading frame 3
Figure 3.3 (a) - (f) show flow chart for translating the 6 reading frames of the non-coding regions.

**Figure 3.3 (d): Flow chart for generating reverse reading frame 1**

- Start the index i from the 1st nucleotide on the complementary non-coding sequence
- Read (i+1)th and (i+2)th nucleotides
- Check if at least one "+" sign in present in the triplet
- Translate the nucleotide in the ith, (i+1)th and (i+2)th positions to the corresponding amino acid
- Move the translation window by 3 nucleotides and assign the start position of the new window to i
- End of sequence?
  - Yes: End of translation
  - No: Check if at least one "+" sign in present in the triplet

**Figure 3.3 (e): Flow chart for generating reverse reading frame 2**

- Start the index i from the 2nd nucleotide on the complementary non-coding sequence
- Read (i+1)th and (i+2)th nucleotides
- Check if at least one "+" sign in present in the triplet
- Translate the nucleotide in the ith, (i+1)th and (i+2)th positions to the corresponding amino acid
- Move the translation window by 3 nucleotides and assign the start position of the new window to i
- End of sequence?
  - Yes: End of translation
  - No: Check if at least one "+" sign in present in the triplet

**Figure 3.3 (f): Flow chart for generating reverse reading frame 3**

- Start the index i from the 3rd nucleotide on the complementary non-coding sequence
- Read (i+1)th and (i+2)th nucleotides
- Check if at least one "+" sign in present in the triplet
- Translate the nucleotide in the ith, (i+1)th and (i+2)th positions to the corresponding amino acid
- Move the translation window by 3 nucleotides and assign the start position of the new window to i
- End of sequence?
  - Yes: End of translation
  - No: Check if at least one "+" sign in present in the triplet

Figure 3.3: Translating the non-coding regions- Figure 3.3 (a) - (f) show flow chart for translating the 6 reading frames of the non-coding regions
3.3 Normalizing the Dipeptide count:

Dipeptide is a pair of amino acids occurring side by side in a translated sequence. Every amino acid can form a dipeptide with any of the other 19 amino acids or with itself. Thus, the total number of possible combination of di-amino acid pairs is $20 \times 20 = 400$.

Our task is to identify and count dipeptides. Upon detecting a dipeptide, we move the detecting window by one amino acid. Thus the second amino acid in the previous dipeptide would be the first amino acid of the succeeding dipeptide. Figure 3.4 shows the detection of dipeptides when window is moved from one position to another. This process is continued until the window reaches the end of the translated sequence.

Figure 3.4: Detecting dipeptides- Example of detecting dipeptides by moving the detecting window
Dipeptides are normalized based on the first amino acid in the dipeptide. The occurrences of all the dipeptides with the same starting (first) amino acid are added together. Therefore this total has the sum of 20 dipeptides. The occurrence of every dipeptide of the group is divided by this sum to normalize the dipeptide occurrence. Normalizing ensures the integrity of the interrelated data. Normalizing the dipeptides based on the first amino acid helps in achieving a higher numerical value compared to overall normalized value. This facilitates us to carry on the further calculations with ease. Figure 3.5 shows the flow chart for calculating and normalizing all the 400 dipeptide occurrences.

The non-coding region is counted in the same way as the coding region. The only difference is that the non-coding regions are analyzed in all six reading frames. The normalized values of respective dipeptide of all the six reading frames are added and divided by 6 to obtain an average normalized value of all the 400 dipeptide occurrences in non-coding region.

Figure 3.6 show the graphs of normalized values of all the dipeptide in coding region versus non-coding region in the *E.coli* genome. We wish to consider how differences in the occurrence of a dipeptide can be used as a distinguishing factor for identifying a coding region in the genome.
Figure 3. 5: Normalizing the dipeptide counts- Flowchart for normalizing dipeptide counts

- Create a window of two amino acids at the beginning of the translated sequence
- Identify the dipeptide
- Increment the respective dipeptide count
- Move the window by one amino acid
- End of translation
- Add all dipeptide counts with common starting amino acid to form group total
- Divide each of the dipeptide count with the respective group total to normalize the dipeptide occurrence

Dipeptide starting with A | Occurrence in coding region | Occurrence in non-coding region
--- | --- | ---
A-A | 0.100444 | 0.096659
A-C | 0.012328 | 0.028641
A-D | 0.052134 | 0.039315
A-E | 0.060770 | 0.033739
A-F | 0.037419 | 0.040301
A-G | 0.077298 | 0.087751
A-H | 0.019837 | 0.026160
A-I | 0.062590 | 0.054098
A-K | 0.041751 | 0.035846
A-L | 0.119344 | 0.081484
A-M | 0.054274 | 0.018553
A-N | 0.034191 | 0.032448
A-P | 0.036984 | 0.053943
A-Q | 0.046718 | 0.033654
A-R | 0.055583 | 0.084566
A-S | 0.056356 | 0.091427
A-T | 0.050557 | 0.057776
A-V | 0.070173 | 0.066926
A-W | 0.017015 | 0.016521
A-Y | 0.021097 | 0.020192
TOTAL | 1.000000 | 1.000000

Figure 3. 6: Example of Normalized occurrence table- Normalized occurrence of dipeptide starting with A, in coding and non-coding regions
3.4: Determining significance of difference:

Our goal is to test if the difference between the normalized values of a given dipeptide in coding and non-coding regions is significant. Finding dipeptides with significant difference of occurrences in coding and non-coding regions potential application in gene recognition as distinguishing elements identify the probable coding regions of a genome. The first 20 genes and 20 equivalent length segments of non-coding region (only one reading frame considered, since all the 6 reading frames are equivalently represent non-coding region) are used as samples for the t-test. Using these samples, the $\bar{x}_{\text{coding}}$ (sample mean of coding region), $\bar{x}_{\text{non-coding}}$ (sample mean of non-coding region), $s_{\text{coding}}$ (sample variance of coding region), $s_{\text{non-coding}}$ (sample variance of non-coding region), $s$ (combined standard deviation) and $t$ are calculated for each dipeptide. Now to test the null-hypothesis ($h_0$) that there is no significant difference between coding region and non-coding region (i.e. between $\mu_{\text{coding}}$ and $\mu_{\text{non-coding}}$) for each dipeptide, $t_{\text{calculated}}$ of each of the dipeptide is compared with the corresponding $t_{\text{tabled}}$ value. If $|t_{\text{calculated}}| > t_{\text{table}}$ at $n+m-2$ degree of freedom ($n$ and $m =$ number of samples of coding and non-coding regions = 20) with 0.05 level of significance, then null hypothesis (hypothesis of no significance) is rejected.

Listed below in Figure 3.7 are the 312 dipeptide with significant difference in their occurrence in coding and non-coding regions:
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>C-A</td>
<td>D-H</td>
<td>E-P</td>
<td>F-V</td>
<td>H-E</td>
<td>L-E</td>
<td>M-I</td>
<td>N-Q</td>
<td>P-Y</td>
<td>T-C</td>
<td>V-I</td>
<td>W-Q</td>
<td>Y-Y</td>
</tr>
<tr>
<td>C-C</td>
<td>D-I</td>
<td>E-Q</td>
<td>F-W</td>
<td>H-F</td>
<td>L-F</td>
<td>M-K</td>
<td>N-R</td>
<td>Q-A</td>
<td>T-D</td>
<td>V-K</td>
<td>W-R</td>
<td></td>
</tr>
<tr>
<td>C-D</td>
<td>D-K</td>
<td>E-R</td>
<td>F-Y</td>
<td>H-G</td>
<td>L-G</td>
<td>M-L</td>
<td>N-S</td>
<td>Q-C</td>
<td>T-E</td>
<td>V-L</td>
<td>W-S</td>
<td></td>
</tr>
<tr>
<td>C-F</td>
<td>D-M</td>
<td>E-T</td>
<td>G-C</td>
<td>H-I</td>
<td>L-I</td>
<td>M-N</td>
<td>N-V</td>
<td>Q-E</td>
<td>T-G</td>
<td>V-N</td>
<td>W-V</td>
<td></td>
</tr>
<tr>
<td>C-G</td>
<td>D-N</td>
<td>E-V</td>
<td>G-D</td>
<td>H-K</td>
<td>L-K</td>
<td>M-P</td>
<td>N-W</td>
<td>Q-F</td>
<td>T-H</td>
<td>V-P</td>
<td>W-W</td>
<td></td>
</tr>
<tr>
<td>C-I</td>
<td>D-Q</td>
<td>E-Y</td>
<td>G-F</td>
<td>H-M</td>
<td>L-M</td>
<td>M-R</td>
<td>P-A</td>
<td>Q-H</td>
<td>T-K</td>
<td>V-R</td>
<td>Y-A</td>
<td></td>
</tr>
<tr>
<td>C-K</td>
<td>D-R</td>
<td>F-A</td>
<td>G-G</td>
<td>H-N</td>
<td>L-N</td>
<td>M-S</td>
<td>P-C</td>
<td>Q-I</td>
<td>T-L</td>
<td>V-S</td>
<td>Y-C</td>
<td></td>
</tr>
<tr>
<td>C-L</td>
<td>D-S</td>
<td>F-C</td>
<td>G-H</td>
<td>H-P</td>
<td>L-P</td>
<td>M-T</td>
<td>P-D</td>
<td>Q-K</td>
<td>T-M</td>
<td>V-T</td>
<td>Y-D</td>
<td></td>
</tr>
<tr>
<td>C-M</td>
<td>D-T</td>
<td>F-D</td>
<td>G-I</td>
<td>H-Q</td>
<td>L-Q</td>
<td>M-V</td>
<td>P-E</td>
<td>Q-L</td>
<td>T-N</td>
<td>V-V</td>
<td>Y-E</td>
<td></td>
</tr>
<tr>
<td>C-N</td>
<td>D-V</td>
<td>F-E</td>
<td>G-K</td>
<td>H-R</td>
<td>L-R</td>
<td>M-W</td>
<td>P-F</td>
<td>Q-M</td>
<td>T-P</td>
<td>V-W</td>
<td>Y-F</td>
<td></td>
</tr>
<tr>
<td>C-P</td>
<td>D-W</td>
<td>F-F</td>
<td>G-L</td>
<td>H-S</td>
<td>L-S</td>
<td>M-Y</td>
<td>P-G</td>
<td>Q-N</td>
<td>T-Q</td>
<td>V-Y</td>
<td>Y-G</td>
<td></td>
</tr>
<tr>
<td>C-Q</td>
<td>D-Y</td>
<td>F-G</td>
<td>G-M</td>
<td>H-T</td>
<td>L-T</td>
<td>N-A</td>
<td>P-H</td>
<td>Q-P</td>
<td>T-R</td>
<td>W-A</td>
<td>Y-H</td>
<td></td>
</tr>
<tr>
<td>C-R</td>
<td>E-A</td>
<td>F-H</td>
<td>G-N</td>
<td>H-V</td>
<td>L-V</td>
<td>N-C</td>
<td>P-I</td>
<td>Q-Q</td>
<td>T-S</td>
<td>W-C</td>
<td>Y-I</td>
<td></td>
</tr>
<tr>
<td>C-S</td>
<td>E-C</td>
<td>F-I</td>
<td>G-P</td>
<td>H-W</td>
<td>L-W</td>
<td>N-D</td>
<td>P-K</td>
<td>Q-R</td>
<td>T-T</td>
<td>W-D</td>
<td>Y-K</td>
<td></td>
</tr>
<tr>
<td>C-T</td>
<td>E-D</td>
<td>F-K</td>
<td>G-Q</td>
<td>H-Y</td>
<td>L-Y</td>
<td>N-E</td>
<td>P-L</td>
<td>Q-S</td>
<td>T-V</td>
<td>W-E</td>
<td>Y-L</td>
<td></td>
</tr>
<tr>
<td>C-V</td>
<td>E-E</td>
<td>F-L</td>
<td>G-R</td>
<td>I-G</td>
<td>M-A</td>
<td>N-F</td>
<td>P-M</td>
<td>Q-T</td>
<td>T-W</td>
<td>W-F</td>
<td>Y-M</td>
<td></td>
</tr>
<tr>
<td>C-W</td>
<td>E-F</td>
<td>F-M</td>
<td>G-S</td>
<td>I-L</td>
<td>M-C</td>
<td>N-G</td>
<td>P-N</td>
<td>Q-V</td>
<td>T-Y</td>
<td>W-G</td>
<td>Y-N</td>
<td></td>
</tr>
<tr>
<td>D-C</td>
<td>E-I</td>
<td>F-Q</td>
<td>G-W</td>
<td>K-S</td>
<td>M-F</td>
<td>N-K</td>
<td>P-R</td>
<td>R-E</td>
<td>V-D</td>
<td>W-K</td>
<td>Y-R</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 3.7:** Significant Dipeptides- Dipeptide with significant difference in their occurrence in coding and non-coding regions

42
Each of the 400 t-tests is conducted with a 5% level of significance. The level of significance is also called the probability of type-I error which signifies the probability of rejecting \( h_0 \) when \( h_0 \) is actually true (Gupta and Kapoor, 1980). This 5% level of significance is indicative of the error margin with which we accept our t-test results. This means that there is a 5 percent chance of getting \( t_{\text{calculated}} \) greater than \( t_{\text{tabled}} \) even though the null hypothesis is true. Thus we accept the differences in normalized values of the above dipeptide occurrence to be significant due to 5% random noise. In other words we accept our results with 95% level of confidence.

Since we are performing multiple t-tests, we consider Bonferroni correction. According to Bonferroni’s correction we expect overall error rate for the set of all t-tests to be 5%. This implies that among the 400 t-tests we could expect 20 false positives i.e. 20 t-test results may wrongly predict that there is a significant difference in coding and non-coding regions. Our experimental results show that there are 312 dipeptides with significant differences of occurrences in coding and non-coding regions. This number is greater than the expected false positives for the 400 t-test. This suggests that coding and non-coding regions can indeed be differentiated based on the normalized values of occurrences of certain dipeptides.

Each of the above tabled 312 dipeptides can be used to differentiate coding region from non-coding. In Section 3.5 we use few of these dipeptides to calculate the false positive and false negative in finding coding regions.
3.5 Distinguishing coding and non-coding regions based on Threshold method

Section 3.5 describes an algorithm for identifying coding and non-coding regions. This section is divided into two subsections. In subsection 3.5.1 we discuss the method used to calculate the threshold to distinguish the values of coding and non-coding regions occurrences. In section 3.5.2 we present our algorithm for predicting the coding and non-coding regions.

3.5.1 Threshold calculation:

The average of the overall normalized occurrences of a dipeptide in coding and non-coding regions is calculated. This average is used as a threshold for differentiating overall normalized coding and non-coding occurrences of a dipeptide. A genomic region is classified based on whether its overall normalized dipeptide occurrence is close to the value determined for coding or non-coding.

When a string of amino acids is considered, the normalized occurrence value of a dipeptide of interest is calculated. This calculated value is compared against the threshold value and depending on what side of the threshold the calculated value lies, the string is classified as coding or non-coding region.
Figure 3.8 shows an example of calculating the threshold of dipeptide C-E.

<table>
<thead>
<tr>
<th>Coding Occurrence of C-E</th>
<th>=</th>
<th>0.057418</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-coding occurrence of C-E</td>
<td>=</td>
<td>0.020380</td>
</tr>
<tr>
<td>Threshold = [(0.057418+0.020380)/2] = 0.038899</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

All occurrence values greater than 0.038899 are considered to be coding.

Figure 3.8: Threshold calculation- An example for calculating the threshold of dipeptide C-E

### 3.5.2 Coding and non-coding region classification

An average length of a gene is arbitrarily assumed to be 500 amino acids long. Starting on the 250\(^{th}\) amino acid in the list of coding sequences, 250 amino acids on either side of this amino acid are considered. The normalized value of occurrence for this string (of length 500 amino acids) is calculated for a given dipeptide. The calculated valued is compared against the
threshold to check whether the string being tested, belongs to coding region of non-coding region. The string is represented by a “c” if it is found to be belonging to coding region else an “n” if it is found to be belonging to non-coding region. This representation is written into a string of characters. The window of 500 amino acids is moved upstream by one amino acid; again the normalized occurrence for that dipeptide is calculated and based on the threshold the new string is classified as “c” or “n” which is concatenated to the growing string of characters. This process is repeated until the window reaches the last amino acid in the list of coding sequences. In the end, the string of characters contains only c’s and n’s. The size of the string of characters is equal to size of list of coding sequences minus 250 amino acids. We follow the steps to generate a string of c’s and n’s for the list of non-coding sequences.

3.6 Results

The definitions of type-1 and type-2 errors depend on the manner in which the null hypothesis is defined. Considering the test is to detect coding region; type-1 error occurs when the algorithm wrongly detects a non-coding region to be coding and type-2 error occurs when the algorithm wrongly detects a coding region to be non-coding. Figure 3.9 shows an example of results for calculating type-1 and type-2 errors of dipeptide C-R. Experimental prediction of overall coding regions based on occurrence of dipeptide C-R is 76.412 % and 23.588% is wrongly predicted as...
non-coding region. Experimental prediction of overall non-coding regions is 59.939% and 40.061% of actual non-coding region is predicted as coding.

Figure 3.9: Calculating error rate- Calculating type 1 and type 2 errors for predicting dipeptide C-R

Table 3.1 shows the Type 1 and Type 2 errors in first 20 significant dipeptides.
Table 3.1: Dipeptides and respective Type 1 and Type 2 errors- Type 1 and Type 2 errors in first 20 significant dipeptides

3.7 Rejection of Threshold method for differentiating coding and non-regions

The performance of the algorithm presented in Section 3.6 is based on the values for type-1 and type-2 errors. Lower values of type-1 and type-2 errors imply a better performance of the algorithm for a given dipeptide. The threshold algorithm is tested for 20 dipeptides that show significant difference in occurrence in coding and non-coding regions. The results show higher
values of type-1 and type-2 errors compared to GeneMarkS gene prediction tool (Besemer et al., 2001). Such values are unacceptable. Figure 3.10 shows and example of such an unacceptable experimental result.

![Experimental Predictions based on occurrence of di-peptide C-T](image)

**Figure 3.10:** Unacceptable Type 1 and Type 2 error values- Experimental performance of threshold for predicting coding and non-coding regions based on the occurrence of dipeptide C-T.

Referring to the Figure 3.10, the algorithm predicts 46.8328 % of type 1 error and 76.1331 % of type 2 error. This means that for the dipeptide C-T, the algorithm wrongly predicts 46.8328 % of actual non-coding region as coding region and 76.1331 % of actual coding region as non-coding region. The dipeptide C-T wrongly predicts a significant portion of coding region as non-coding region.
and vice versa. Hence, it is not a good discriminator of coding and non-coding regions based on threshold method. Experimental results for most of the dipeptides test show higher values of type 1 and type 2 errors.

The gene predicting algorithms like Genescan and Genescan++ have accuracy over 80% (Bernal et al., 2007). We would have accepted the method selecting the coding and non-coding regions based on normalized values of occurrences if the type 1 error was less than 25%. Thus the method of classifying a given region as coding or as non-coding based on the threshold of the normalized values of a single dipeptide (with significant difference in occurrence) is rejected.
Chapter 4

Identification of coding regions based on frequency distribution

This chapter describes the frequency distribution method for differentiating coding and non-coding regions. Section 4.1.1 details the procedure followed in developing the frequency distribution for all the dipeptides that show significant difference of occurrence in coding and non-coding regions. Section 4.1.2 describes the selection of dipeptides that are potential identifiers of coding regions. Section 4.2 explains the methods for ranking randomly selected coding and non-coding strings of E.coli’s genome, based on the probability of number of dipeptides identifying these strings as coding regions. Section 4.3 calculates the type 1 and type 2 errors in identifying coding and non-coding regions of E.coli’s genome and validates the results of E.coli’s genome on that of Salmonella’s genome.

4.1.1 Frequency distribution patterns

The C++ code that generates the data for frequency distribution, randomly selects 1000 strings of amino acids from coding region (from list of coding sequences) and from non-coding region (from non-coding.txt). Each of these strings consists of 500 amino acids. The dipeptide
occurrence for every string is calculated. Depending on the maximum and minimum value of the dipeptide occurrence, 20 bins are created, where each bin is of size

\[
\frac{(\text{max } \text{dipeptide occurrence}) - (\text{min } \text{dipeptide occurrence})}{20}
\]

This step ensures that the dipeptide occurrence for all the 2000 strings (1000 from coding and 1000 from non-coding regions) is combined into 20 groups. When the dipeptide occurrence value for a string of amino acid is generated, the value would fall into one of the 20 bins. The frequency distribution for a dipeptide is generated by counting the number of strings of coding and non-coding regions contained in each of the 20 bins. A coding weight of calculated for each of the bin. The coding weight is defined as the ratio of the number of coding strings in the bin to the sum of all strings from coding and non-coding regions in the bin. The complete frequency distribution of each significant dipeptide is written into a ‘.csv’ file so that it will be beneficial for further manipulation using MS-EXCEL. Finally, the sum of all the coding weights is calculated for each of the dipeptide whose frequency distribution is generated and is outputted to ‘CodingWeights.csv’ file. Figure 4.1 shows the flowchart for generating frequency distribution for a dipeptide with significant difference of occurrence in coding and non-coding regions.
Read “coding.txt” and “transl.txt”

Randomly select 1000 strings of amino acids from each of the files

Calculate dipeptide occurrence of string from coding region and store in the array: ‘CodingVal[1001]’

Calculate dipeptide occurrences for all string from non-coding region and store in array: ‘NonCodingVal[1001]’

Calculate the max and min amongst ‘CodingVal’ and ‘NonCodingVal’

Bin size = \( \frac{\text{max} - \text{min}}{20} \)

Initialize
premin = min;
pemax = min + bin size;
i = 0;

Is \( \text{premin} < (\text{CodingVal}[j] \text{ or } \text{NonCodingVal}[j]) < \text{premax} \)?

Yes

Increment the respective bin count

j++

No

Is \( j < 1000 \) ?

Yes

Coding Weight

\[ = \frac{\#\text{Coding strings}}{\#\text{Coding strings} + \#\text{NonCoding strings}} \]

No

START

Initialize \( j=0 \)

B

A
Figure 4.1: Generating Frequency Distribution—Flowchart showing steps followed in generating frequency distribution for a given dipeptide.

A

premin = premax;
premax = premax + bin size;
i++;

B

Yes

Is i < 20?

No

Cumulative coding weight = \sum coding weights

End of frequency
Table 4.1 shows an example of the frequency distribution for the dipeptide A-M.

<table>
<thead>
<tr>
<th>Range</th>
<th>Coding Values</th>
<th>Non-Coding Values</th>
<th>Coding Likelihood</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 - 0.004255</td>
<td>196</td>
<td>585</td>
<td>0.25096</td>
</tr>
<tr>
<td>0.004255 - 0.008511</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0.008511 - 0.012766</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0.012766 - 0.017021</td>
<td>69</td>
<td>14</td>
<td>0.831325</td>
</tr>
<tr>
<td>0.017021 - 0.021277</td>
<td>124</td>
<td>72</td>
<td>0.632653</td>
</tr>
<tr>
<td>0.021277 - 0.025532</td>
<td>143</td>
<td>112</td>
<td>0.560784</td>
</tr>
<tr>
<td>0.025532 - 0.029787</td>
<td>61</td>
<td>60</td>
<td>0.504132</td>
</tr>
<tr>
<td>0.029787 - 0.034043</td>
<td>35</td>
<td>48</td>
<td>0.421687</td>
</tr>
<tr>
<td>0.034043 - 0.038298</td>
<td>54</td>
<td>28</td>
<td>0.658537</td>
</tr>
<tr>
<td>0.038298 - 0.042553</td>
<td>59</td>
<td>20</td>
<td>0.746835</td>
</tr>
<tr>
<td>0.042553 - 0.046809</td>
<td>32</td>
<td>25</td>
<td>0.561404</td>
</tr>
<tr>
<td>0.046809 - 0.051064</td>
<td>36</td>
<td>4</td>
<td>0.9</td>
</tr>
<tr>
<td>0.051064 - 0.055319</td>
<td>30</td>
<td>7</td>
<td>0.810811</td>
</tr>
<tr>
<td>0.055319 - 0.059575</td>
<td>28</td>
<td>6</td>
<td>0.823529</td>
</tr>
<tr>
<td>0.059575 - 0.063833</td>
<td>11</td>
<td>1</td>
<td>0.916667</td>
</tr>
<tr>
<td>0.06383 - 0.068085</td>
<td>15</td>
<td>3</td>
<td>0.833333</td>
</tr>
<tr>
<td>0.068085 - 0.072340</td>
<td>25</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0.072340 - 0.076596</td>
<td>20</td>
<td>3</td>
<td>0.869565</td>
</tr>
<tr>
<td>0.076596 - 0.080851</td>
<td>6</td>
<td>5</td>
<td>0.545455</td>
</tr>
<tr>
<td>0.080851 - 0.085106</td>
<td>6</td>
<td>6</td>
<td>0.5</td>
</tr>
</tbody>
</table>

This column represents the number of strings from coding region that's normalized occurrence values fall in the range of the respective bin.

This column represents the number of strings from coding region that's normalized occurrence values fall in the range of the respective bin.

This column represents the respective coding weights for every bin.

The 20 bins

Table 4.1: Frequency distribution table- Example of Frequency distribution table generated for dipeptide A-M.
4.1.2 Selecting the discriminating dipeptides

In this section we focus on steps for selecting dipeptides that are potential identifiers of coding regions. Based on the value of cumulative weight, a dipeptide is considered as a potential identifier of coding. Table 4.2 shows the frequency distribution of the dipeptide Q-G which is a potential coding region identifier.

<table>
<thead>
<tr>
<th>Range</th>
<th>Coding Values</th>
<th>Non-Coding Values</th>
<th>Coding Weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 - 0.0166667</td>
<td>225</td>
<td>530</td>
<td>0.298013</td>
</tr>
<tr>
<td>0.0166667 - 0.033333</td>
<td>49</td>
<td>30</td>
<td>0.620253</td>
</tr>
<tr>
<td>0.0333333 - 0.05</td>
<td>80</td>
<td>119</td>
<td>0.40201</td>
</tr>
<tr>
<td>0.05 - 0.0666667</td>
<td>75</td>
<td>78</td>
<td>0.490196</td>
</tr>
<tr>
<td>0.0666667 - 0.083333</td>
<td>71</td>
<td>92</td>
<td>0.435583</td>
</tr>
<tr>
<td>0.0833333 - 0.1</td>
<td>38</td>
<td>50</td>
<td>0.431818</td>
</tr>
<tr>
<td>0.1 - 0.116667</td>
<td>106</td>
<td>41</td>
<td>0.721088</td>
</tr>
<tr>
<td>0.116667 - 0.133333</td>
<td>98</td>
<td>28</td>
<td>0.777778</td>
</tr>
<tr>
<td>0.133333 - 0.15</td>
<td>60</td>
<td>12</td>
<td>0.833333</td>
</tr>
<tr>
<td>0.15 - 0.166667</td>
<td>48</td>
<td>4</td>
<td>0.923077</td>
</tr>
<tr>
<td>0.166667 - 0.183333</td>
<td>42</td>
<td>1</td>
<td>0.976744</td>
</tr>
<tr>
<td>0.183333 - 0.2</td>
<td>15</td>
<td>4</td>
<td>0.789474</td>
</tr>
<tr>
<td>0.2 - 0.216667</td>
<td>27</td>
<td>7</td>
<td>0.794118</td>
</tr>
<tr>
<td>0.216667 - 0.233333</td>
<td>16</td>
<td>1</td>
<td>0.941176</td>
</tr>
<tr>
<td>0.233333 - 0.25</td>
<td>15</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0.25 - 0.266667</td>
<td>10</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0.266667 - 0.283333</td>
<td>8</td>
<td>1</td>
<td>0.888889</td>
</tr>
<tr>
<td>0.283333 - 0.3</td>
<td>5</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0.3 - 0.316667</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0.316667 - 0.333333</td>
<td>7</td>
<td>2</td>
<td>0.777778</td>
</tr>
</tbody>
</table>

Table 4.2: Example of potential coding region identifying dipeptide- Frequency distribution of normalized occurrences of dipeptide Q-R
The dipeptide Q-R has 8 bins whose coding weights are greater than 0.8. This implies that 80% of the strings in these bins are coding. In other words, when the normalized occurrence value of a translated genomic string occurs within any of these bins, the extent of it being considered a coding region is more than 80%. The 0.8 cut off value of coding weight was chosen after rejecting the cut off value of 0.85. At 0.85 cut off value, only a few dipeptides have bins with coding weights equal to or greater than this value. Since 0.8 cut off value presents satisfactory results, smaller cut off values are not tried.

The dipeptide Q-R has the highest cumulative coding weight. Dipeptides with higher cumulative coding weights are considered to be better identifiers of coding regions.

Table 4.3 shows the frequency distribution of dipeptide T-G which is not a potential coding identifier. Referring to Table 4.3 the coding weights of none of the bins is greater than 80%, the cumulative weight is 6.45109, which is very low compared. Thus T-G is not a potential identifier of coding region.

The dipeptides, whose frequency distributions are generated, are arranged in descending order of the cumulative coding weights. In section 4.2, the top 100 dipeptides of this list are used as the voters for considering a translated genomic string as a coding region.
Table 4.3: Example of a dipeptide which is a non-potential coding identifier: Frequency distribution of normalized occurrences of dipeptide T-G

<table>
<thead>
<tr>
<th>Range</th>
<th>Coding Values</th>
<th>Non-Coding Values</th>
<th>Coding Weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 - 0.0166667</td>
<td>118</td>
<td>76</td>
<td>0.608247</td>
</tr>
<tr>
<td>0.0166667 - 0.033333</td>
<td>16</td>
<td>56</td>
<td>0.222222</td>
</tr>
<tr>
<td>0.033333 - 0.05</td>
<td>153</td>
<td>123</td>
<td>0.554348</td>
</tr>
<tr>
<td>0.05 - 0.0666667</td>
<td>88</td>
<td>85</td>
<td>0.508671</td>
</tr>
<tr>
<td>0.0666667 - 0.083333</td>
<td>107</td>
<td>95</td>
<td>0.529703</td>
</tr>
<tr>
<td>0.083333 - 0.1</td>
<td>135</td>
<td>129</td>
<td>0.511364</td>
</tr>
<tr>
<td>0.1 - 0.116667</td>
<td>114</td>
<td>79</td>
<td>0.590674</td>
</tr>
<tr>
<td>0.116667 - 0.133333</td>
<td>102</td>
<td>68</td>
<td>0.6</td>
</tr>
<tr>
<td>0.133333 - 0.15</td>
<td>58</td>
<td>58</td>
<td>0.5</td>
</tr>
<tr>
<td>0.15 - 0.166667</td>
<td>32</td>
<td>68</td>
<td>0.32</td>
</tr>
<tr>
<td>0.166667 - 0.183333</td>
<td>32</td>
<td>47</td>
<td>0.405063</td>
</tr>
<tr>
<td>0.183333 - 0.2</td>
<td>17</td>
<td>15</td>
<td>0.53125</td>
</tr>
<tr>
<td>0.2 - 0.216667</td>
<td>26</td>
<td>30</td>
<td>0.464286</td>
</tr>
<tr>
<td>0.216667 - 0.233333</td>
<td>2</td>
<td>17</td>
<td>0.105263</td>
</tr>
<tr>
<td>0.233333 - 0.25</td>
<td>0</td>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td>0.25 - 0.266667</td>
<td>0</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>0.266667 - 0.283333</td>
<td>0</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>0.283333 - 0.3</td>
<td>0</td>
<td>16</td>
<td>0</td>
</tr>
<tr>
<td>0.3 - 0.316667</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0.316667 - 0.333333</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

4.2 Ranking the translated genomic sequences

This section describes the methods for ranking randomly selected coding and non-coding strings, based on the extent of being identified as coding regions.

In order to rank the translated genomic sequences, the top one hundred dipeptides are selected from the sorted list of dipeptides. For a given translated genomic sequence, the normalized occurrence values for all the 100 dipeptides are calculated. The calculated normalized values are
placed in the appropriate bins and assigned the respective coding weights of the bins. The translated genomic string is considered as a coding region only if, the coding normalized occurrence value of a dipeptide is in the bin that has a coding weight greater than or equal to 0.8.

The assigned coding weight for the normalized occurrence of a dipeptide signifies the likelihood of identifying a translated genomic sequence as coding region. The coding weight of 0.8 is considered as the threshold for identifying the coding regions. A string is considered as a coding region only if, the normalized occurrence value of a dipeptide is in the bin that has a coding weight equal to or greater than 0.8.

The confidence level of identifying a coding region directly depends on the number of dipeptides identifying a translated genomic sequence as coding region. The coding (list of coding sequences) and non-coding (list of non-coding sequences) regions are read and randomly 200 strings from each of the regions are randomly selected. Each of these selected strings has a length of 500 amino acids. Further, the number of such dipeptide identifiers for every randomly selected string of coding and non-coding regions is counted. All the coding and non-coding strings and their respective count dipeptide identifiers are outputted to two separate files. These two files are arranged in the descending order of the number of dipeptides identifier. Figure 4.2 shows the flowchart for ranking the randomly selected coding and non-coding strings.
Start

Read ‘Coding.txt’

From “TextFile1.txt” read the top 100 dipeptide identifiers to dipeptide[100]

A

GenomicStringCounter = 0

Is GenomicStringCounter = 0 ?

YES

Randomly select a translated genomic string of length 500 amino acids

dipeptideCounter = 0;
NumOfDipeptideIdentifiers = 0;

NO

GenomicStringCounter++

B

Is dipeptideCounter < 100?

YES

Output the string and respective NumOfDipeptideIdentifiers to “SortedCodingList.csv” file

C

D

No

Yes
Figure 4.2: Ranking the genomic strings- Flowchart for ranking the randomly selected coding and non-coding strings.
Table 4.4 shows the number of genomic strings that are identified as coding regions by various dipeptide identifiers.

<table>
<thead>
<tr>
<th>Coding Region</th>
<th>Non-coding Region</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Strings</td>
<td>Number of dipeptide coding identifiers</td>
</tr>
<tr>
<td>1</td>
<td>18</td>
</tr>
<tr>
<td>1</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>15</td>
</tr>
<tr>
<td>6</td>
<td>14</td>
</tr>
<tr>
<td>14</td>
<td>13</td>
</tr>
<tr>
<td>19</td>
<td>12</td>
</tr>
<tr>
<td>20</td>
<td>11</td>
</tr>
<tr>
<td>17</td>
<td>10</td>
</tr>
<tr>
<td>24</td>
<td>9</td>
</tr>
<tr>
<td>35</td>
<td>8</td>
</tr>
<tr>
<td>15</td>
<td>7</td>
</tr>
<tr>
<td>20</td>
<td>6</td>
</tr>
<tr>
<td>19</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 4. 4: Ranking the genomic stings- Number of genomic strings of Escherichia coli 536 that are identified as coding regions by various dipeptide identifiers.

The Table 4.4 is generated by referring to “SortedCodingList.csv” and “SortedNoncodingList.csv” files. The entries of the table are obtained by counting the number of translated genomic strings having the same number of dipeptide identifiers for both coding and non-coding regions.

Referring to the entries of coding regions, number of dipeptide coding identifiers ranges from 18 to 3. Higher number of dipeptide identifiers is indicative of a higher confidence level in accepting the given translated genomic string as coding.
Considering the first row of the coding region of the Table 4.4, 18 dipeptides identify a coding string as coding. This implies that 18 dipeptides are at least 80% confident in characterizing this string as coding.

For the non-coding regions, the number of dipeptide coding identifiers is expected to be closed to zero. Since the likelihood of accepting a given translated genomic sequence to be coding is not 100%, certain false positives are also expected. This means that there is a possibility of characterizing a non-coding string wrongly as coding. From the table 4.4 it can be noticed that number of dipeptide coding identifiers of non-coding region ranges from 5 to 0.

Based on the above observations, it is possible to establish a criterion for identifying the coding regions of *E.coli*’s genome. A translated genomic string can be declared as coding if the number of dipeptide coding identifiers is equal to or greater than 6. The confidence level of acceptance increases with the increase in the number dipeptide coding identifiers. If the number of dipeptide coding identifiers ranges between 0 and 2; the translated genomic string is more likely to be non-coding. The lower number of dipeptide coding identifiers signifies a more likelihood of a translated genomic string to be non-coding. If the number of dipeptide identifiers of a translated genomic string is in between 5 and 3; the genomic string could be either coding or non-coding.

In the next section, the findings of the *E.coli*’s genome are validated on Salmonella-typhi’s genome.
4.3 Results

In Section 4.2, a threshold of 6 dipeptide identifiers is derived for discriminating coding and non-coding regions. The accuracy of discriminating the coding and non-coding regions based on frequency distribution method is tested on 400 randomly selected translated strings from coding and non-coding regions (200 strings from coding and 200 strings from non-coding regions). The experimental methods are similar to that in Section 4.2. A given translated genomic region is identified as coding region if the number of dipeptide identifiers is more than 5.

The results for identifying coding and non-coding regions based on this threshold are tabulated in Table 4.5. Referring to the Table 4.5, 96.5% of the coding sequences are correctly identified as coding and all the translated non-coding strings are correctly classified as non-coding. The type 1 and type 2 errors of 3.5% and 0% of frequency distribution method is definitely better than that of normalized value method.

This section also validates the results of *E.coli*’s genome on salmonella’s genome. The focus is to determine if the frequency distribution data and the threshold value of *E.coli* could facilitate in identifying coding and non-coding regions of salmonella. This query is evaluated by identifying the coding and non-coding regions of salmonella’s genome using the same set of ranked dipeptides, their respective frequency distributions and the threshold used for *E.coli*’s genome.

The experimentation methods for calculating the type 1 and type 2 errors of *E.coli*’s genome are conducted on salmonella’s genome. The results of this experimentation are shown in Table 4.6. Table 4.6 shows that 79.5% of coding strings of salmonella’s genome are correctly identified as
coding and all the translated strings of non-coding region is correctly identified. These results prove that the coding and non-coding regions of a non-decoded genome can be identified with a high accuracy based on the genomic knowledge of its genetically related organism.

Table 4.5: Type 1 and Type 2 errors in E.coli- Type 1 and Type 2 errors for predicting coding and non-coding region of E.coli’s genome using a threshold of 6 dipeptide identifiers

Table 4.6: Type 1 and Type 2 errors in salmonella- Type 1 and Type 2 errors for predicting coding and non-coding region of E.coli’s genome using a threshold of 6 dipeptide identifiers
Chapter 5

Conclusion

Biology is all about probing into the nature. Bio-informatics is one of the powerful tools to answer the biological questions. Living systems are similar to computers. Computer data contains 1’s and 0’s and biological data contains nucleotide bases which are A, G, T and C’s.

There is a pattern in many random processes; may it be ripples in a sand dune or schools of fish, a visible pattern is evident. This thesis tries to find patterns in genomic data. Our work tries to address the questions; “Can we differentiate coding and non-coding regions based on dipeptide usage?” “Can we detect coding regions using dipeptide occurrence?”

5.1 Contribution

The prime contribution of this thesis is to facilitate a gene detecting tool to search for probable coding regions of a genome. This thesis presents a simple and quick method to find the likelihood of a genomic sequence to be coding region. The research is targeted on E.coli’s genome whose results are validated with that of the salmonella’s genome.
5.1.1 T-test

T-tests are performed to examine the significant differences between the coding and non-coding regions of *E.coli*’s genome. The experimental results of t-test on *E.coli* and salmonella’s genome exceed the number of false positives. Thus it can be asserted with confidence that differentiation of coding and non-coding regions can be effected based on the dipeptide usage.

5.1.2. Detecting the coding regions

The experimental results of this research demonstrate the likelihood of identifying the coding region based on the number of dipeptide identifiers. The likelihood of identifying the coding region is higher if the number of dipeptide coding identifiers is more.

Due to an acceptance confidence level of 80% for each dipeptide identifier, a few false positive dipeptide identifiers are also expected. The experimental results show a threshold of 6 dipeptide coding identifiers for *E.coli*’s genome. If the number of dipeptide coding identifiers of a translated genomic string is more than 6, then it is likely to be coding. Based on this threshold value, type 1 and type 2 errors for identifying coding and non-coding regions are calculated. The coding regions are correctly identified to accuracy of 96.5% and for that of non-coding regions, it is 100%. 
The ranked dipeptides, their respective frequency distribution and the threshold of *E.coli*’s genome are utilized to identify coding and non-coding regions of salmonella’s genome. 79.5% of coding regions are correctly identified and the entire non-coding regions are correctly identified as non-coding.

The validation of the results of *E.coli* genome on the salmonella’s genome proves that the obtained results are not due to the randomness of the nature. This quick and simple procedure facilitates a gene detecting tool to search for probable coding regions in the non-coding region of a non decoded genome with the knowledge of its genetically related genome.

### 5.2 Comparison of accuracies of various gene predictors

The Table 5.1 compares the accuracies of our algorithm with some of the available prokaryotic gene finding methods. Refering to Table 5.1, all the programs other than CRITICA use the data set from the *E.coli* genome, and CRITICA uses the data set from *S.typhimurium* genome. The accuracy of our algorithm in correctly identifying coding region is slightly lower than those of *GenMark.hmm*- 6-LBA, 4-4BA, 5-5BA, *MetaGene*, *MetaGene Annotator* and CRITICA, but is slightly better than those of the remaining models of *GeneMark.hmm*. However, the accuracy of
our algorithm in correctly identifying the non-coding region out scores those of all the other programs listed in the Table 5.1.

<table>
<thead>
<tr>
<th>Program</th>
<th>Model</th>
<th>Sensitivity (in %)</th>
<th>Specificity (in %)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GeneMark.hmm</td>
<td>HAL-99</td>
<td>94.93</td>
<td>93.38</td>
</tr>
<tr>
<td></td>
<td>C-3BA</td>
<td>96.24</td>
<td>94.8</td>
</tr>
<tr>
<td></td>
<td>C-3MT</td>
<td>96.32</td>
<td>93.31</td>
</tr>
<tr>
<td></td>
<td>C-MBA</td>
<td>96.34</td>
<td>93.31</td>
</tr>
<tr>
<td></td>
<td>3-3BA</td>
<td>95.64</td>
<td>93.85</td>
</tr>
<tr>
<td></td>
<td>3-LBA</td>
<td>95.97</td>
<td>93.77</td>
</tr>
<tr>
<td></td>
<td>4-4BA</td>
<td>96.7</td>
<td>94.57</td>
</tr>
<tr>
<td></td>
<td>5-5BA</td>
<td>96.75</td>
<td>94.66</td>
</tr>
<tr>
<td></td>
<td>6-6BA</td>
<td>96.49</td>
<td>94.77</td>
</tr>
<tr>
<td></td>
<td>6-LBA</td>
<td>96.99</td>
<td>94.63</td>
</tr>
<tr>
<td>MetaGene</td>
<td></td>
<td>97.22</td>
<td>91.08</td>
</tr>
<tr>
<td>MetaGene Annotator</td>
<td></td>
<td>97.15</td>
<td>92.35</td>
</tr>
<tr>
<td>CRITICA</td>
<td></td>
<td>98.74</td>
<td>99.12</td>
</tr>
<tr>
<td>Our Algorithm</td>
<td></td>
<td>96.5</td>
<td>100</td>
</tr>
</tbody>
</table>

Table 5.1: Accuracy of gene prediction: A comparison of accuracies of various gene finding methods (Source: Zhu et al., 2010 and Bader and Olsen, 1999)
5.3 Future Work

*E. coli* and Salmonella are prokaryotes. Validating the finding of this research on Eukaryotes will open up new vistas for further research. In eukaryotes, a larger percentage of the genome is comprised of junk DNA. The junk DNA is comprised of many types of non-coding DNA sequences that have known biological functions. Finding a criterion to identify such biologically functioning non-coding regions based on dipeptide usage will be very beneficial. An immediate offshoot of such a discovery could be to determine the tendency of genes transforming to pseudo genes and vice versa.

Investigating the usage of the common dipeptide identifiers for a group of related species, will reduce the efforts of our algorithm in directing a gene detecting tool toward probable coding regions of a non-decoded genome.
Bibliography


APPENDIX A: C++ Program to separate and translate coding region

/* This program reads the *E.coli* genome, counts the dipeptide occurrences and normalises the occurrences based on the second aminoacid in the dipeptide. Finally, the program generates 20 .CSV files wherein, in the first column it writes the normalised dipeptide values and in the second column the count of the 2nd dipeptide. */

```cpp
#include<iostream>
#include<conio.h>
#include<stdio.h>
#include<string.h>
#include <fstream>
#include<iomanip>
using namespace std;
float a[20][20], cnt=0, tmp=0;
float A=0, C=0, D=0, E=0, F=0, G=0, H=0, I=0, K=0, L=0, M=0, N=0, P=0, Q=0, R=0, S=0, T=0, V=0, W=0, Y=0;
void aminoCount(char s[], int len);

int main()
{
   for(int i=0; i<20; i++)
      for(int j=0; j<20; j++)
         a[i][j]=0;

   char *sequence;
   int sequenceLength = 0;
   int sequenceArraySize = 0;
   char str[200];
   char filename[100] = "E.coli_genome.txt";
   char file_number[50];
   char ch[401]; //, junk[200]; // ch is for valuable data,
   char s[20][300]; // st is used for parts of the family

   //////////////////////////////////////////////////////
   sequence = (char*)malloc(CHUNK * sizeof(char));
   sequenceArraySize = CHUNK;

   ifstream in(filename); // for reading X.NCBI36.part1.fai
```

78
while(!in.eof())
{
    in.getline(ch,400);
    sequenceLength =0;
    char* pch= NULL;
    int j=0;
    pch = strtok (ch," /=\n");
    while (pch != NULL)
    {
        strcpy(s[j],pch);
        pch = strtok (NULL, " /=\n");
        j++;
    }
    if(strcmp(s[0],"translation")==0)
    {
        if(s[1][strlen(s[1])-1]=='"') //if the translation ends on the same line.
        {
            for(int i=1;i<strlen(s[1])-1; i++)
            {
                if (sequenceLength >= sequenceArraySize)
                {
                    sequenceArraySize += CHUNK;
                }
                sequence[sequenceLength] = s[1][i];
                sequenceLength++;
            }
            aminoCount( sequence, sequenceLength);
        }
        else //for multiple line width translation
        {
            for(int i=1 ;i<strlen(s[1]);i++)
            {
                if (sequenceLength >= sequenceArraySize)
                {
                    sequenceArraySize += CHUNK;
                }
                sequence[sequenceLength] = s[1][i];
                sequenceLength++;
           }
        }
        in.getline(ch,400);
        char* pch=NULL;
        j=0;
        pch = strtok (ch," /=\n");
        while (pch != NULL)
        {
            strcpy(s[j],pch);
            pch = strtok (NULL, " /=\n");
            j++;
        }
    }
}
while(s[0][strlen(s[0])-1]!=""
{
    for(int i=0 ;i<strlen(s[0]);i++)
    {
        if (sequenceLength >= sequenceArraySize)
        {
            sequenceArraySize += CHUNK;
            sequence = (char *) realloc(sequence, sequenceArraySize*sizeof(char));
        }
        sequence[sequenceLength] = s[0][i];
        sequenceLength++;
    }
    in.getline(ch,400);
    char* pch;
    j=0;
    pch = strtok (ch," /=\n");
    while (pch != NULL)
    {
        strcpy(s[j],pch);
        pch = strtok (NULL, " /=\n");
        j++;
    }
}
//end of while(s[0][strlen(s[0])-1]!=""
for(int i=0 ;i<strlen(s[0])-1 ; i++)
{
    if (sequenceLength >= sequenceArraySize)
    {
        sequenceArraySize += CHUNK;
        sequence = (char *) realloc(sequence, sequenceArraySize*sizeof(char));
    }
    sequence[sequenceLength] = s[0][i];
    sequenceLength++;
}
aminoCount( sequence, sequenceLength);
//end of if(strcmp(s[0],"translation=")==0)
//end of while( !in.eof())

>Description of function aminoCount
************************************
in.close();
ofstream out0("A.csv",ios::app);
tmp=0;//tmp store total # of A's in the entire gene
for(int i=0;i<20;i++)
{
    tmp +=a[0][i];
}

80
cout<<"A-A"<<","<<a[0][0]/tmp<<",""<<A/cnt"<<",""<<endl; //cnt has the total # of coding nucleotides
cout<<"A-C"<<","<<a[0][1]/tmp<<",""<<C/cnt"<<",""<<endl;
cout<<"A-D"<<","<<a[0][2]/tmp<<",""<<D/cnt"<<",""<<endl;
cout<<"A-E"<<","<<a[0][3]/tmp<<",""<<E/cnt"<<",""<<endl;
cout<<"A-F"<<","<<a[0][4]/tmp<<",""<<F/cnt"<<",""<<endl;
cout<<"A-G"<<","<<a[0][5]/tmp<<",""<<G/cnt"<<",""<<endl;
cout<<"A-H"<<","<<a[0][6]/tmp<<",""<<H/cnt"<<",""<<endl;
cout<<"A-I"<<","<<a[0][7]/tmp<<",""<<I/cnt"<<",""<<endl;
cout<<"A-K"<<","<<a[0][8]/tmp<<",""<<K/cnt"<<",""<<endl;
cout<<"A-L"<<","<<a[0][9]/tmp<<",""<<L/cnt"<<",""<<endl;
cout<<"A-M"<<","<<a[0][10]/tmp<<",""<<M/cnt"<<",""<<endl;
cout<<"A-N"<<","<<a[0][11]/tmp<<",""<<N/cnt"<<",""<<endl;
cout<<"A-P"<<","<<a[0][12]/tmp<<",""<<P/cnt"<<",""<<endl;
cout<<"A-Q"<<","<<a[0][13]/tmp<<",""<<Q/cnt"<<",""<<endl;
cout<<"A-R"<<","<<a[0][14]/tmp<<",""<<R/cnt"<<",""<<endl;
cout<<"A-S"<<","<<a[0][15]/tmp<<",""<<S/cnt"<<",""<<endl;
cout<<"A-T"<<","<<a[0][16]/tmp<<",""<<T/cnt"<<",""<<endl;
cout<<"A-V"<<","<<a[0][17]/tmp<<",""<<V/cnt"<<",""<<endl;
cout<<"A-W"<<","<<a[0][18]/tmp<<",""<<W/cnt"<<",""<<endl;
cout<<"A-Y"<<","<<a[0][19]/tmp<<",""<<Y/cnt"<<",""<<endl;
cout.close();
ofstream out1("C.csv",ios::app);
tmp=0;
for(int i=0;i<20;i++)
{
    tmp +=a[1][i];
}

out1<<"C-A"<<","<<a[1][0]/tmp<<",""<<A/cnt"<<",""<<endl;
out1<<"C-C"<<","<<a[1][1]/tmp<<",""<<C/cnt"<<",""<<endl;
out1<<"C-D"<<","<<a[1][2]/tmp<<",""<<D/cnt"<<",""<<endl;
out1<<"C-E"<<","<<a[1][3]/tmp<<",""<<E/cnt"<<",""<<endl;
out1<<"C-F"<<","<<a[1][4]/tmp<<",""<<F/cnt"<<",""<<endl;
out1<<"C-G"<<","<<a[1][5]/tmp<<",""<<G/cnt"<<",""<<endl;
out1<<"C-H"<<","<<a[1][6]/tmp<<",""<<H/cnt"<<",""<<endl;
out1<<"C-I"<<","<<a[1][7]/tmp<<",""<<I/cnt"<<",""<<endl;
out1<<"C-K"<<","<<a[1][8]/tmp<<",""<<K/cnt"<<",""<<endl;
out1<<"C-L"<<","<<a[1][9]/tmp<<",""<<L/cnt"<<",""<<endl;
out1<<"C-M"<<","<<a[1][10]/tmp<<",""<<M/cnt"<<",""<<endl;
out1<<"C-N"<<","<<a[1][11]/tmp<<",""<<N/cnt"<<",""<<endl;
out1<<"C-P"<<","<<a[1][12]/tmp<<",""<<P/cnt"<<",""<<endl;
out1<<"C-Q"<<","<<a[1][13]/tmp<<",""<<Q/cnt"<<",""<<endl;
out1<<"C-R"<<","<<a[1][14]/tmp<<",""<<R/cnt"<<",""<<endl;
out1<<"C-S"<<","<<a[1][15]/tmp<<",""<<S/cnt"<<",""<<endl;
out1<<"C-T"<<","<<a[1][16]/tmp<<",""<<T/cnt"<<",""<<endl;
out1<<"C-V"<<","<<a[1][17]/tmp<<",""<<V/cnt"<<",""<<endl;
out1<<"C-W"<<","<<a[1][18]/tmp<<",""<<W/cnt"<<",""<<endl;
out1<<"C-Y"<<","<<a[1][19]/tmp<<",""<<Y/cnt"<<",""<<endl;
out1.close();
ofstream out2("D.csv",ios::app);
tmp=0;
for(int i=0;i<20;i++)
{  
    tmp +=a[2][i];
}
out2<<" D-A"<<","<<a[2][0]/tmp<<","<<A/cnt"<<","<<endl;
out2<<" D-C"<<","<<a[2][1]/tmp<<","<<C/cnt"<<","<<endl;
out2<<" D-D"<<","<<a[2][2]/tmp<<","<<D/cnt"<<","<<endl;
out2<<" D-E"<<","<<a[2][3]/tmp<<","<<E/cnt"<<","<<endl;
out2<<" D-F"<<","<<a[2][4]/tmp<<","<<F/cnt"<<","<<endl;
out2<<" D-G"<<","<<a[2][5]/tmp<<","<<G/cnt"<<","<<endl;
out2<<" D-H"<<","<<a[2][6]/tmp<<","<<H/cnt"<<","<<endl;
out2<<" D-I"<<","<<a[2][7]/tmp<<","<<I/cnt"<<","<<endl;
out2<<" D-K"<<","<<a[2][8]/tmp<<","<<K/cnt"<<","<<endl;
out2<<" D-L"<<","<<a[2][9]/tmp<<","<<L/cnt"<<","<<endl;
out2<<" D-M"<<","<<a[2][10]/tmp<<","<<M/cnt"<<","<<endl;
out2<<" D-N"<<","<<a[2][11]/tmp<<","<<N/cnt"<<","<<endl;
out2<<" D-P"<<","<<a[2][12]/tmp<<","<<P/cnt"<<","<<endl;
out2<<" D-Q"<<","<<a[2][13]/tmp<<","<<Q/cnt"<<","<<endl;
out2<<" D-R"<<","<<a[2][14]/tmp<<","<<R/cnt"<<","<<endl;
out2<<" D-S"<<","<<a[2][15]/tmp<<","<<S/cnt"<<","<<endl;
out2<<" D-T"<<","<<a[2][16]/tmp<<","<<T/cnt"<<","<<endl;
out2<<" D-V"<<","<<a[2][17]/tmp<<","<<V/cnt"<<","<<endl;
out2<<" D-W"<<","<<a[2][18]/tmp<<","<<W/cnt"<<","<<endl;
out2<<" D-Y"<<","<<a[2][19]/tmp<<","<<Y/cnt"<<","<<endl;
out2<<" D-Y"<<","<<a[2][20]/tmp<<","<<Y/cnt"<<","<<endl;
out2.close();
} //end of main

return 0;
} //end of main

void aminoCount(char s[],int len)
{
    cnt+=len;
    for(int i=0;i<len-1;i++)
    {
        if(s[i]=='A')A++;
        else if(s[i]=='C')C++;
        else if(s[i]=='D')D++;
        else if(s[i]=='E')E++;
        else if(s[i]=='F')F++;
        else if(s[i]=='G')G++;
        else if(s[i]=='H')H++;
        else if(s[i]=='I')I++;
        else if(s[i]=='K')K++;
        else if(s[i]=='L')L++;
        else if(s[i]=='M')M++;
        else if(s[i]=='N')N++;
        else if(s[i]=='P')P++;
        else if(s[i]=='Q')Q++;
        else if(s[i]=='R')R++;
    }
}
else if(s[i]=='S')S++;
else if(s[i]=='T')T++;
else if(s[i]=='V')V++;
else if(s[i]=='W')W++;
else if(s[i]=='Y')Y++;
}

for(int i=0;i<len-1;i++)
{
    if(s[i]=='A')
    {
        if(s[i+1]=='A')a[0][0]++;
        else if(s[i+1]=='C')a[0][1]++;
        else if(s[i+1]=='D')a[0][2]++;
        else if(s[i+1]=='E')a[0][3]++;
        else if(s[i+1]=='F')a[0][4]++;
        else if(s[i+1]=='G')a[0][5]++;
        else if(s[i+1]=='H')a[0][6]++;
        else if(s[i+1]=='I')a[0][7]++;
        else if(s[i+1]=='K')a[0][8]++;
        else if(s[i+1]=='L')a[0][9]++;
        else if(s[i+1]=='M')a[0][10]++;
        else if(s[i+1]=='N')a[0][11]++;
        else if(s[i+1]=='P')a[0][12]++;
        else if(s[i+1]=='Q')a[0][13]++;
        else if(s[i+1]=='R')a[0][14]++;
        else if(s[i+1]=='S')a[0][15]++;
        else if(s[i+1]=='T')a[0][16]++;
        else if(s[i+1]=='V')a[0][17]++;
        else if(s[i+1]=='W')a[0][18]++;
        else if(s[i+1]=='Y')a[0][19]++;
    }
    else if(s[i]=='C')
    {
        if(s[i+1]=='A')a[1][0]++;
        else if(s[i+1]=='C')a[1][1]++;
        else if(s[i+1]=='D')a[1][2]++;
        else if(s[i+1]=='E')a[1][3]++;
        else if(s[i+1]=='F')a[1][4]++;
        else if(s[i+1]=='G')a[1][5]++;
        else if(s[i+1]=='H')a[1][6]++;
        else if(s[i+1]=='I')a[1][7]++;
        else if(s[i+1]=='K')a[1][8]++;
        else if(s[i+1]=='L')a[1][9]++;
        else if(s[i+1]=='M')a[1][10]++;
        else if(s[i+1]=='N')a[1][11]++;
        else if(s[i+1]=='P')a[1][12]++;
        else if(s[i+1]=='Q')a[1][13]++;
        else if(s[i+1]=='R')a[1][14]++;
        else if(s[i+1]=='S')a[1][15]++;
        else if(s[i+1]=='T')a[1][16]++;
        else if(s[i+1]=='V')a[1][17]++;
        else if(s[i+1]=='W')a[1][18]++;
        else if(s[i+1]=='Y')a[1][19]++;
    }
}
else if(s[i+1]=='Q') a[1][13]++;
else if(s[i+1]=='R') a[1][14]++;
else if(s[i+1]=='S') a[1][15]++;
else if(s[i+1]=='T') a[1][16]++;
else if(s[i+1]=='V') a[1][17]++;
else if(s[i+1]=='W') a[1][18]++;
else if(s[i+1]=='Y') a[1][19]++;
}
تسمى هذه العملية على النحو نفسه ل残د من 18 المجموعات

/*********************/

//same is repeated for rest of 18 groups/
/*********************/

}
APPENDIX B: C++ Program to separate and translate non-coding region

/* This program reads the E.coli genome, separates and translates the non-coding regions into 3 reading frames in the forward direction.*/

#include<iostream>
#include<conio.h>
#include<stdio.h>
#include<string.h>
#include <fstream>
#include<iomanip>
using namespace std;

int main()
{
    char *sequence;
    long sequenceLength =0;
    long sequenceArraySize =0;
    char nextChar;
    char str[200];
    int lim=0;
    char filename[30]="NC_008253.ppt"; //consists of starting and ending locations of genes
    char datafile[30]="NC_008253.fna"; //consists of the entire genome
    char outfile[30]="NoncodingE.coli.txt"; //to be concatenated later
    char outfile0[30]="translation0.txt";
    char outfile1[30]="translation1.txt";
    char outfile2[30]="translation2.txt";
    char ch[400];// ch stores valuable dat,
    long count=0;
    char f1[400]="",f2[400]="";
    char st[40][50]; //st is used for parts of the family
    int x=0,j=0,i=0,k=0,a=0,b=0;
    long start =0, end=0;
    int stopCount0=0, stopCount1=0, stopCount2=0;
    int start0=0, start1=0, start2=0;
    ofstream out(outfile,ios::app);
    ofstream out0(outfile0,ios::app);
    ofstream out1(outfile1,ios::app);
    ofstream out2(outfile2,ios::app);
    ifstream in(filename);//"NC_008253.ppt" file set as input file.
    //This file has the start and stop pos of genes
    ifstream file_op(datafile);//"NC_008253.fna" file set as input file.
    //This file has the entire genome
    sequence=(char*)malloc(CHUNK * sizeof(char));
    sequenceArraySize = CHUNK;
    for(i=0;i<1;i++)
    {
file_op.getline(str,200);//elimination of the first line from "NC_008253.ptt"
cout<<str<<endl;
}
for(i=0;i<3;i++)
{
in.getline(str,200);//elimination of the first 3 lines from "NC_008253.fna"
cout<<str<<endl;
}
in.getline(ch,200);
char * pch;
j=0, start=0, end=0;
pch = strtok(ch," .");
while (pch != NULL)
{
    strcpy(st[j],pch);
pch = strtok(NULL," .");
j++;
}
start = atol(st[0]);
start = start-1;
end = atol(st[1]);
end = end-1;
while(!file_op.eof())
{
    file_op.get(nextChar);
    if ((nextChar !=EOF)&&(nextChar !='\n'))
    {
        if(count==start)
        {
            for(count=start;count<=end;count++)
            {
                if(sequenceLength >= sequenceArraySize)
                {
                    sequenceArraySize += CHUNK;
                    sequence = (char *)realloc(sequence, sequenceArraySize*sizeof(char));
                }
                sequence[sequenceLength] = '+';
                sequenceLength++;
                file_op.get(nextChar);
            }
    }if(!in.eof())
    {
        in.getline(ch,400);
        char * pch;
j=0;
pch = strtok(ch," .");
    while (pch != NULL)
    {
        strcpy(st[j],pch);
pch = strtok(NULL," .");
j++;
    }
    start = atol(st[0]);
    start = start-1;
if (start <= end)
{
    start = end + 1;
}
cout << start << endl;
end = atol(st[1]);
end = end - 1;
}
else
{
    if (sequenceLength >= sequenceArraySize)
    {
        sequenceArraySize += CHUNK;
        sequence = (char *) realloc(sequence, sequenceArraySize * sizeof(char));
    }
    if (nextChar == 'T')
        sequence[sequenceLength] = 'U';
    else
        sequence[sequenceLength] = nextChar;
    sequenceLength++;
    count++;
}
}
sequence[sequenceLength] = '\0';
out << sequence;
cout << "sequence over" << endl;
cout << "count" << count << "", "sequenceLength" << sequenceLength << endl;

/**************************BEGINNING OF THE TRANSLATION******************************/
for (i = 0; i < sequenceLength; i++)
out << sequence[i];
for (i = 0; i < sequenceLength - 2; i += 3)
{
    if (sequence[i] == '+' || sequence[i + 1] == '+' || sequence[i + 2] == '+')
    {
    }
    if (sequence[i] == 'A')
    {
        if (sequence[i + 1] == 'A')
        {
            if (sequence[i + 2] == 'A')
            {
                out << 'K';
            }
            else if (sequence[i + 2] == 'C')
            {
                out << 'N';
            }
            else if (sequence[i + 2] == 'G')
            {
                out << 'K';
            }
            else if (sequence[i + 2] == 'U')
            {
                out << 'U';
            }
        }
    }
```c
} else if (sequence[i+1] == 'C') {
    if (sequence[i+2] == 'A')
        out0 << 'T';
    else if (sequence[i+2] == 'C')
        out0 << 'T';
    else if (sequence[i+2] == 'G')
        out0 << 'T';
    else if (sequence[i+2] == 'U')
        out0 << 'T';
}
else if (sequence[i+1] == 'G') {
    if (sequence[i+2] == 'A')
        out0 << 'R';
    else if (sequence[i+2] == 'C')
        out0 << 'S';
    else if (sequence[i+2] == 'G')
        out0 << 'R';
    else if (sequence[i+2] == 'U')
        out0 << 'S';
}
else if (sequence[i+1] == 'U') {
    if (sequence[i+2] == 'A')
        out0 << 'I';
    else if (sequence[i+2] == 'C')
        out0 << 'I';
    else if (sequence[i+2] == 'G')
        out0 << 'M';
    start1++;
    } else if (sequence[i+2] == 'U')
        out0 << 'I';

} /*same is repeated for the rest of
18 groups*/
```

out2.close();
return 0;
} //end of main
APPENDIX C: C++ Program for T-test

/* This program reads the files ‘twenty_genes.txt’ and ‘twenty_nongenes.txt’ that consist of first 20 translated genes and 20 equivalent length non-coding strings of E.coli genome respectively. 2 sample t-test is conducted using the translated strings contained in the above mentioned files, as the data set. The t-test is conducted with 95% confidence*/

#include<iostream>
#include<conio.h>
#include<stdio.h>
#include<string.h>
#include <fstream>
#include<iomanip>
#include<math.h>
using namespace std;
float a[20][20],tmp=0;
float h0=0,h1=0,h2=0,h3=0;
float aa=0;

void aminoCount(char s[]);
void pri (int f, int g, float tCalculated);
ofstream out1("significant output.txt",ios::app);
int main()
{
    char str[20][20][100];
    char filename[100]="twenty_genes.txt";
    char filenamenc[100]="twenty_nongenes.txt";
    float gn[20][20][20];
    float ngn[20][20][20];
    char ch[2001];//,junk[200];// ch is for valuable dat,
    char ch1[2001];
    char c1='\0',c2='\0';
    float ugn=0,ungn=0;
    float sgn=0,sngn=0;
    float s=0;
    float t[20][20];

    ofstream out("output.txt",ios::app);
    ifstream in(filename);//for reading twenty_genes.txt
    int i=0;
    while( !in.eof())
    {
        in.getline(ch,2000);
        if(isalpha(ch[0]))
        {
            aminoCount(ch);//compute the amino acid pair count
            for(int j=0;j<20;j++)
                for(int k=0;k<20;k++)
                    float ugn=0,ungn=0;
                    float sgn=0,sngn=0;
                    float s=0;
                    float t[20][20];

                    ofstream out("output.txt",ios::app);
                    ifstream in(filename);//for reading twenty_genes.txt
                    int i=0;
                    while( !in.eof())
                    {
                        in.getline(ch,2000);
                        if(isalpha(ch[0]))
                        {
                            aminoCount(ch);//compute the amino acid pair count
                            for(int j=0;j<20;j++)
                                for(int k=0;k<20;k++)
                                    
                                    89
{  
gn[i][j][k]=a[j][k];  
}

i++;  
}
}
ifstream in1(filenamenc);//for reading X.NCBI36.part1.fa.aln
i=0;
while( !in1.eof())  
{cout<<"hi dude"<<endl;
in1.getline(ch1,2000);
if(isalpha(ch1[0]))  
{  
aminoCount(ch1);//compute the dipeptide count
    
    for(int j=0;j<20;j++)  
        
    for(int k=0;k<20;k++)  
    {  
        ngn[i][j][k]=a[j][k];  
    }  
    i++;  
}

/****************************
**********************************************/
/*Calculation of mean of ngn and gn*/
for(int j=0;j<20;j++)  
    for(int k=0;k<20;k++)  
    {  
        ugn = 0;  
        ungn = 0;  
        for(i=0;i<20;i++)  
        {  
            ugn += gn[i][j][k];  
            ungn += ngn[i][j][k];  
        }  
        ugn /= 20;  
        ungn /= 20;
    }  

/****************************
**********************************************/
/*Calculation of s1 and s2*/
sgn = 0;  
sngn = 0;  
for(i=0;i<20;i++)  
{  
    sgn += pow((gn[i][j][k]-ugn),2);  
    sngn += pow((ngn[i][j][k]-ungn),2);  
}  
sgn /=19;  //calculation of s1 and s2 squares  
sngn /=19;
s = (19*(sgn + sgn))/38; //calculation of s
s = sqrt(s);

/////////////////////////////////////////////////

\[ t[j][k] = \frac{\text{fabs}(ugn - ungn)}{(s * 0.316228)} \] //t's value calculated

if (t[j][k] <= 2.024) //tabled t-value for 18 degree of freedom with .05 level of significance
    strcpy(str[j][k], "no significant difference");
else
    {\n        aa++; //keeps count of amino acid pairs with significant differences
        strcpy(str[j][k], "difference is significant ");
        pri(j,k,t[j][k]); //prints significant amino acids
    }
}

for (int f=0; f<20; f++)
{
    if (f==0)c1 = 'A';
    else if (f==1)c1 = 'C';
    else if (f==2)c1 = 'D';
    else if (f==3)c1 = 'E';
    else if (f==4)c1 = 'F';
    else if (f==5)c1 = 'G';
    else if (f==6)c1 = 'H';
    else if (f==7)c1 = 'I';
    else if (f==8)c1 = 'K';
    else if (f==9)c1 = 'L';
    else if (f==10)c1 = 'M';
    else if (f==11)c1 = 'N';
    else if (f==12)c1 = 'P';
    else if (f==13)c1 = 'Q';
    else if (f==14)c1 = 'R';
    else if (f==15)c1 = 'S';
    else if (f==16)c1 = 'T';
    else if (f==17)c1 = 'V';
    else if (f==18)c1 = 'W';
    else if (f==19)c1 = 'Y';

    for (int g=0; g<20; g++)
    {\n        if (g==0)c2 = 'A';
        else if (g==1)c2 = 'C';
        else if (g==2)c2 = 'D';
        else if (g==3)c2 = 'E';
        else if (g==4)c2 = 'F';
        else if (g==5)c2 = 'G';
        else if (g==6)c2 = 'H';
        else if (g==7)c2 = 'I';
        else if (g==8)c2 = 'K';
        else if (g==9)c2 = 'L';
        else if (g==10)c2 = 'M';
        else if (g==11)c2 = 'N';
        else if (g==12)c2 = 'P';
        else if (g==13)c2 = 'Q';
        else if (g==14)c2 = 'R';
        else if (g==15)c2 = 'S';
    }
else if(g==16)c2='T';
else if(g==17)c2='V';
else if(g==18)c2='W';
else if(g==19)c2='Y';
out<<c1<<"-"<<c2<<":"<<str[f][g]<<endl;
}
out<<"**********"<<endl;

/* Calculating the normalised amino acid pairs with significant differences*/

h0=h0/aa;
h1=h1/aa;
h2=h2/aa;
h3=h3/aa;
ofstream outt("nomalisedT-test.txt",ios::app);
outt<<"Hydrophobic -Hydrophobic = "<<h0<<endl;
outt<<"Hydrophobic -Hydrophillic = "<<h1<<endl;
outt<<"Hydrophillic -Hydrophobic = "<<h2<<endl;
outt<<"Hydrophillic -Hydrophillic = "<<h3<<endl;
in.close();
return 0;
}/end of main

/********************************************************************************counting starts now********************************************************************************

void aminoCount(char s[])
{
    int len=0;
    len = strlen(s);
    for(int i=0; i<20; i++)
        for(int j=0;j<20;j++)
            a[i][j]=0;

    for(int i=0;i<len;i++)
    {
        if(s[i]=='A')
        {
            if(s[i+1]=='A')a[0][0]++;
            else if(s[i+1]=='C')a[0][1]++;
            else if(s[i+1]=='D')a[0][2]++;
            else if(s[i+1]=='E')a[0][3]++;
            else if(s[i+1]=='F')a[0][4]++;
            else if(s[i+1]=='G')a[0][5]++;
            else if(s[i+1]=='H')a[0][6]++;
            else if(s[i+1]=='I')a[0][7]++;
            else if(s[i+1]=='K')a[0][8]++;
            else if(s[i+1]=='L')a[0][9]++;
            else if(s[i+1]=='M')a[0][10]++;
            else if(s[i+1]=='N')a[0][11]++;
            else if(s[i+1]=='P')a[0][12]++;
            else if(s[i+1]=='Q')a[0][13]++;
            else if(s[i+1]=='R')a[0][14]++;
            else if(s[i+1]=='S')a[0][15]++;
            else if(s[i+1]=='T')a[0][16]++;
            else if(s[i+1]=='V')a[0][17]++;
            else if(s[i+1]=='W')a[0][18]++;
            else if(s[i+1]=='Y')a[0][19]++;
        }
    }
}
else if(s[i+1]=='T') a[0][16]++;
else if(s[i+1]=='V') a[0][17]++;
else if(s[i+1]=='W') a[0][18]++;
else if(s[i+1]=='Y') a[0][19]++;
}
else if(s[i]=='C')
{
if(s[i+1]=='A') a[1][0]++;
else if(s[i+1]=='C') a[1][1]++;
else if(s[i+1]=='D') a[1][2]++;
else if(s[i+1]=='E') a[1][3]++;
else if(s[i+1]=='F') a[1][4]++;
else if(s[i+1]=='G') a[1][5]++;
else if(s[i+1]=='H') a[1][6]++;
else if(s[i+1]=='I') a[1][7]++;
else if(s[i+1]=='K') a[1][8]++;
else if(s[i+1]=='L') a[1][9]++;
else if(s[i+1]=='M') a[1][10]++;
else if(s[i+1]=='N') a[1][11]++;
else if(s[i+1]=='P') a[1][12]++;
else if(s[i+1]=='Q') a[1][13]++;
else if(s[i+1]=='R') a[1][14]++;
else if(s[i+1]=='S') a[1][15]++;
else if(s[i+1]=='T') a[1][16]++;
else if(s[i+1]=='V') a[1][17]++;
else if(s[i+1]=='W') a[1][18]++;
else if(s[i+1]=='Y') a[1][19]++;
}

/* same is repeated for the rest 18 groups*/

for(int i=0;i<20;i++)
{
    tmp=0;
    for(int j=0;j<20;j++)
    {
        tmp +=a[i][j];
    }
    for(int j=0;j<20;j++)
    {
        if (tmp == 0)
a[i][j]=0;
else
    a[i][j]=a[i][j]/tmp;
}
if(c1=='A'||c1=='C'||c1=='G'||c1=='I'||c1=='L'||c1=='K'||c1=='M'||c1=='F'||c1=='P'||c1=='S'||c1=='W'||c1=='Y'||c1=='V')
    strcpy(s1,"Hydrophobic");
else
    strcpy(s1,"Hydrophillic");
if
(c2=='A'||c2=='C'||c2=='G'||c2=='I'||c2=='L'||c2=='K'||c2=='M'||c2=='F'||c2=='P'||c2=='S'||c2=
='W'||c2=='Y'||c2=='V')
    strcpy(s2,"Hydrophobic");
else
    strcpy(s2,"Hydrophillic");
out1<<c1<<"-"<<c2<<"":"<<s1<<"-"<<s2<<","<<tCalculated<<endl;
if(strcmp(s1,"Hydrophobic")&& strcmp(s2,"Hydrophobic")) h0++;
else if(strcmp(s1,"Hydrophobic")&& strcmp(s2,"Hydrophillic")) h1++;
else if (strcmp(s1,"Hydrophillic")&& strcmp(s2,"Hydrophobic")) h2++;
else if (strcmp(s1,"Hydrophillic")&& strcmp(s2,"Hydrophillic")) h3++;
APPENDIX D: C++ Program for generating frequency distributions of significant dipeptides

/*The following program generates the frequency distribution for dipeptides listed in 'TextFile1.txt' in 20 bins. The program starts by reading 'coding.txt' and 'transl.txt' files. It generates 1000 random strings from each file, determines the normalized occurrence values for all the 2000 strings in 20 bins. Finds the bins with 0.8 or more coding / non-coding weights ratio and lists in a separate column. In the end it finds the cumulative coding weights of all the bins of a dipeptide and lists the dipeptide and the respective cumulative coding weights in 'WeightsOfDipeptides.csv' file. This file could be used to rank and select the top 100 dipeptides for ranking the genomic strings */

#include<iostream>
#include<conio.h>
#include<stdio.h>
#include<math.h>
#include<string.h>
#include <fstream>
#include<iomanip>
using namespace std;

float aminoCount(char s[],int len,char aa1, char aa2);

int main()
{
    char fl_name[50];
    char *sequence;
    int sequenceLength =0;
    int sequenceArraySize =0;
    char filename[100]="transl.txt";
    char file_number[50];
    char  s[501];//st is used for parts of the family
    float val[1001];
    char nextChar;
    int ngACount=0;// keeps track of the # of amino acids equivalent to 20 genes that
                  //are to be eliminated
    /*******************************************************************************/
    sequence=(char*)malloc(CHUNK * sizeof(char));
    sequenceArraySize = CHUNK;
    ifstream in(filename);
    while(!in.eof())
    {
        if (sequenceLength >= sequenceArraySize)
        {
            sequenceArraySize += CHUNK;
            sequence = (char *) realloc(sequence, sequenceArraySize*sizeof(char));
        }
        in.get(nextChar);
        ngACount++;
        sequence[sequenceLength] = nextChar;
        sequenceLength++;
    }
}
/*At this the entire non-coding region is read and is pointed by sequence*/
/*Now the coding region is going to be read*/
char *csequence;
int csequenceLength = 0;
int csequenceArraySize = 0;
char cfilename[100] = "Coding.txt";
char cfile_number[50];
float cval[1001];
char cnextChar;

// declaration of sequence
////////////////////////////////////////////
char* csequence = (char*)malloc(CHUNK * sizeof(char));
csequenceArraySize = CHUNK;
ifstream cdin(cfilename);// for reading X.NCBI36.part1.fa.aln
while(!cdin.eof())
{
    if (csequenceLength >= csequenceArraySize)
    {
        csequenceArraySize += CHUNK;
        csequence = (char *) realloc(csequence, csequenceArraySize * sizeof(char));
    }
    cdin.get(cnextChar);
    csequence[csequenceLength] = cnextChar;
    csequenceLength++;
}
// end of reading the coding region

/******** Start reading the text file containing names for all .csv files to be generated*/

ifstream fl_in("TextFile1.txt"); // This text file has list of dipeptides that are to be tested
ofstream out1("WeightsOfDipeptides.csv"); // the tested dipeptides and respective coding weights are stored in this file
out1<<"Dipeptide","Coding Weights"<<endl;
while (!fl_in.eof())
{
    fl_in.getline(fl_name,3);
    strcat(fl_name,"_Distribution.csv"); // Frequency distribution of individual dipeptides are stored in separate files here
    ofstream out(fl_name,ios::app);
    out<<"Range ","Coding Values","Non-Coding Values"<<","<<"CodingWeights"<<endl;
    int RnC,RC,t=0,Ct=0;
    float wtsum=0;
    for (int i=0; i< 1000; i++)
    {
        t=0;
        Ct=0;
        RnC = rand() % (sequenceLength-500) + 1;
        for(int j=RnC;j<(RnC+500);j++)
        {
            s[t]= sequence[j];
            t++;
        }
    }
}
val[i] = aminoCount(s, 500, f1_name[0], f1_name[1]);//amino count of non-coding part
RC = rand() % (csequenceLength-500) + 1;
for(int j=RC;j<(RC+500);j++)
{
    s[Ct]= csequence[j];
    Ct++;
}
cval[i] = aminoCount(s, 500, f1_name[0], f1_name[1]);

/**************************************************************************************/
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% %*/
/*Finding the max and min of val and cval  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%://%*/
float amin =0, amax= 0;
for(int i=0;i<1000;i++)
{
    /*finding minimum of 3 numbers*/
    if(amin > cval[i])
    {
        amin = cval[i];
        if(amin > val[i])
        {
            amin = val[i];
        }
    }
    else if(amin > val[i])
    {
        amin = val[i];
        if(amin > cval[i])
        {
            amin = cval[i];
        }
    }
    /* %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
    /* finding maximum of 3 numbers*/
    if(amax < cval[i])
    {
        amax = cval[i];
        if(amax < val[i])
        {
            amax = val[i];
        }
    }
    else if(amax < val[i])
    {
        amax = val[i];
        if(amax< cval[i])
    }
float bin = \((\text{amax} - \text{amin})/20\);  
float premin = amin, premax = amin+bin;  
int ccnt, ncnt;  
for (int j=0; j<20; j++)  
{  
    ccnt=0; ncnt =0;  
    float wt=0;  
    for (int i=0; i<1000; i++)  
    {  
        if((cval[i]>= premin)&&(cval[i]<premax))  
        {  
            ccnt++;  
        }  
        if((val[i]>=premin)&&(val[i]<premax))  
        {  
            ncnt++;  
        }  
    }  
    if((ccnt>4 || ncnt>4))  
    {  
        wt = (float)ccnt/(ccnt+ncnt);  
    }  
    {  
        wt = -(float)ncnt/(ccnt+ncnt);  
    }  
    else  
    {  
        wt=0;  
    }  
    out<<setprecision(6)<<premin<< " - "<<setprecision(6)<<premax<<","<<setprecision(6)<<ncnt<<","<<setprecision(6)<<ccnt<<","<<wt<<endl;  
    wtsum = wtsum + wt;  
    premin = premax;  
    premax = premax +bin;  
}  
out1<<fl_name[0]<<fl_name[1]<<","<<wtsum<<endl;  
out.close();  
}
f1_in.close();
out1.close();

return 0;

} //end of main

////////////////////////////////////////////////////////////////////////////////////////////
/* The following part of the code calculates the total occurrence
of A (stored in a) and total occurrence of Q following A (stored in b).
the normalised occurrence is calculated by dividing b/a. This value of b
is then compared with 0.0401. If b>0.0401 it is considered to be a chunk belonging
to coding region else it is considered to be in the non-coding region*/

float aminoCount(char s[],int len,char aa1, char aa2)
{
    int a=0;
    float b=0;
    for(int i=0;i<len-2;i++)
    {
        if(s[i]== aa1)
        {
            a++;
            if(s[i+1]== aa2)b++;
        }
    }
    if(a==0)
        return 0;
    else
    {
        return b/a;
    }
}
APPENDIX E: C++ Program for ranking the genomic strings

/*This program randomly selects 200 translated coding strings and 200 translated non-coding strings. These strings are ranked based on the number of dipeptide identifiers. The strings along with the number of coding identifiers are written into 'SortedCodingList.csv' and 'SortedNonCodingList.csv' files. A threshold if found for identifying the number of coding identifiers. Using this threshold and minor changes int this program, one can find the type-1 & -2 errors in predicting coding and non-coding regions. This idea is used in finding type-1 & -2 errors for both E.coli and salmonella’s genome.*/

#include<iostream>
#include<conio.h>
#include<stdio.h>
#include<math.h>
#include<string.h>
#include <fstream>
#include<iomanip>
using namespace std;

float aminoCount(char s[],int len,char aa1, char aa2);

int main ()
{
    char fl_name[50], fl_name1[50];char* outfl;
    char fl_str[3][50];

    char  val_str[5][100];//char array to store the tokenised values of the ch array
    char  *sequence;
    int sequenceLength =0;
    int sequenceArraySize =0;

    /***********Structure to store the samples and respective weights**********************/
    struct StringSamples
    {
        float weights;
        char strg[500];
    }sam[200], stemp;

    /*******************************************/
    char csequence[CHUNK * sizeof(char)];
    char cfilename[100]="Salmonellatransl.txt";
    char cfile_number[50];
    float cval;
    char cnextChar;
    char s[500];
    char fl_rdr[300]; // stores the lines of .csv file temporarily
    csequence=(char*)malloc(CHUNK * sizeof(char));
csequenceArraySize = CHUNK
ifstream cdin(cfilename);//for reading X.NCBI36.part1.fa.aln
while(!cdin.eof())
{
    if (csequenceLength >= csequenceArraySize)
    {
        csequenceArraySize += CHUNK;
        csequence = (char *) realloc(csequence, csequenceArraySize*sizeof(char));
    }
    cdin.get(cnextChar);
    csequence[csequenceLength] = cnextChar;
    csequenceLength++;
}
//end of reading the coding region
for (int ind=0; ind<200; ind++)
{
    float wt=0;//stores the weight of a sample for all the dipeptides
    int RnC,count=0,CodingCount=0;// count keeps in track of the # of dipeptides and coding
    RnC = rand() % (csequenceLength-500) + 1;//random point selected
    int t=0;
    for(int j=RnC;j<(RnC+500);j++)// random coding sequence selected
    {
        s[t]= csequence[j];
        t++;
    }
    ifstream fl_in("TextFile1.txt");
    while (!fl_in.eof())
    {
        count++;
        fl_in.getline(fl_name,50);//start reading all the frequency distribution files/
        strcpy(fl_name1,fl_name);
        char* fl_pch= NULL;
        int j=0;
        fl_pch = strtok (fl_name,"_");
        while (fl_pch != NULL)
        {
            strcpy(fl_str[j],fl_pch);
            fl_pch = strtok (NULL, "_");
            j++;
        }
        cval = aminoCount( s, 500,fl_str[0][0],fl_str[0][1]);//start reading all the frequencies in the distribution files*/
        ifstream csv_in( fl_name1,ios::app);
        csv_in.getline(fl_rdr, 300);//read the first bin and its frequency of occurrence.
        char* pch= NULL;
        float bn_min=0, bn_max=0,p_wt=0 ;//p_wt stores the calculated proportional wt
        int j=0;
        pch = strtok (fl_rdr,"-.");
        while (pch != NULL)
        {


```c
{
    strcpy(val_str[j], pch);
    pch = strtok (NULL, "-, ");
    j++;
}
bn_min = atof(val_str[0]);
bn_max = atof(val_str[1]);
if(((cval >= bn_min) && (cval < bn_max)) && (atof(val_str[4]) > 0.8 ))
{
    CodingCount++;
}
} // end of forloop for reading a Distribution file
csv_in.close();
} // end of while
strcpy(sam[ind].strg, s);
if (count == 0)
{
    sam[ind].weights = 0;
} else
    sam[ind].weights = CodingCount;
fl_in.close();
} // end of testing all the 200 samples
for (int j=0; j<200; j++)
{
    for(int i=0; i<199; i++)
    {
        if(sam[i+1].weights > sam[i].weights)
        {
            stemp = sam[i];             // swap elements
            sam[i] = sam[i+1];
            sam[i+1] = stemp;
        }
    }
}
int Threshold = 5; // Initialize the threshold
ofstream out("SortedSalmonellaNonCodingList.csv", ios::app);
for(int i=0; i<200; i++)
{
    for(int j=0; j<500; j++)
    {
        out << sam[i].strg[j];
    }
    out << "", << sam[i].weights;
    if (sam[i].weights > Threshold)
        out << "" Coding string" << endl;
    else
        out << "" Non-Coding string" << endl;
}
out.close();
return 0;
```
float aminoCount(char s[],int len,char aa1, char aa2)
{
    int a=0;
    float b=0;
    for(int i=0;i<len-2;i++)
    {
        if(s[i]== aa1)
        {
            a++;
            if(s[i+1]== aa2)b++;
        }
    }
    if(a==0)
        return 0;
    else
    {
        return b/a;
    }
}
APPENDIX F: List of significant dipeptides ranked as per respective cumulative coding weights

<table>
<thead>
<tr>
<th>Dipeptide</th>
<th>Cumulative Coding Weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>QG</td>
<td>14.1013</td>
</tr>
<tr>
<td>LE</td>
<td>13.4347</td>
</tr>
<tr>
<td>LD</td>
<td>13.0615</td>
</tr>
<tr>
<td>FG</td>
<td>13.042</td>
</tr>
<tr>
<td>QA</td>
<td>12.8197</td>
</tr>
<tr>
<td>VE</td>
<td>12.4519</td>
</tr>
<tr>
<td>FE</td>
<td>12.4361</td>
</tr>
<tr>
<td>TM</td>
<td>12.4267</td>
</tr>
<tr>
<td>AM</td>
<td>12.3677</td>
</tr>
<tr>
<td>MD</td>
<td>12.3168</td>
</tr>
<tr>
<td>KE</td>
<td>12.157</td>
</tr>
<tr>
<td>LG</td>
<td>11.911</td>
</tr>
<tr>
<td>QE</td>
<td>11.8657</td>
</tr>
<tr>
<td>VL</td>
<td>11.8647</td>
</tr>
<tr>
<td>VG</td>
<td>11.7745</td>
</tr>
<tr>
<td>VM</td>
<td>11.771</td>
</tr>
<tr>
<td>TE</td>
<td>11.7321</td>
</tr>
<tr>
<td>GM</td>
<td>11.6851</td>
</tr>
<tr>
<td>LV</td>
<td>11.6457</td>
</tr>
<tr>
<td>TL</td>
<td>11.6103</td>
</tr>
<tr>
<td>FA</td>
<td>11.4718</td>
</tr>
<tr>
<td>SE</td>
<td>11.4274</td>
</tr>
<tr>
<td>GD</td>
<td>11.4095</td>
</tr>
<tr>
<td>FD</td>
<td>11.2838</td>
</tr>
<tr>
<td>NG</td>
<td>11.1686</td>
</tr>
<tr>
<td>DD</td>
<td>11.0669</td>
</tr>
<tr>
<td>KL</td>
<td>11.0199</td>
</tr>
<tr>
<td>EQ</td>
<td>10.8931</td>
</tr>
<tr>
<td>QM</td>
<td>10.8824</td>
</tr>
<tr>
<td>YL</td>
<td>10.8256</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dipeptide</th>
<th>Cumulative Coding Weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>YD</td>
<td>10.6636</td>
</tr>
<tr>
<td>PN</td>
<td>10.5888</td>
</tr>
<tr>
<td>PD</td>
<td>10.5875</td>
</tr>
<tr>
<td>PQ</td>
<td>10.5118</td>
</tr>
<tr>
<td>GG</td>
<td>10.4781</td>
</tr>
<tr>
<td>DI</td>
<td>10.4595</td>
</tr>
<tr>
<td>VD</td>
<td>10.2741</td>
</tr>
<tr>
<td>DE</td>
<td>10.2633</td>
</tr>
<tr>
<td>GV</td>
<td>10.2602</td>
</tr>
<tr>
<td>IL</td>
<td>10.2289</td>
</tr>
<tr>
<td>VK</td>
<td>10.2228</td>
</tr>
<tr>
<td>LL</td>
<td>10.2181</td>
</tr>
<tr>
<td>IG</td>
<td>10.1934</td>
</tr>
<tr>
<td>RE</td>
<td>10.1337</td>
</tr>
<tr>
<td>TQ</td>
<td>9.95405</td>
</tr>
<tr>
<td>YQ</td>
<td>9.94819</td>
</tr>
<tr>
<td>LN</td>
<td>9.89243</td>
</tr>
<tr>
<td>HL</td>
<td>9.83026</td>
</tr>
<tr>
<td>EE</td>
<td>9.82804</td>
</tr>
<tr>
<td>VA</td>
<td>9.75682</td>
</tr>
<tr>
<td>ME</td>
<td>9.72057</td>
</tr>
<tr>
<td>GW</td>
<td>9.7193</td>
</tr>
<tr>
<td>MG</td>
<td>9.63285</td>
</tr>
<tr>
<td>NE</td>
<td>9.56614</td>
</tr>
<tr>
<td>PE</td>
<td>9.56495</td>
</tr>
<tr>
<td>NV</td>
<td>9.55195</td>
</tr>
<tr>
<td>VV</td>
<td>9.53359</td>
</tr>
<tr>
<td>PL</td>
<td>9.51742</td>
</tr>
<tr>
<td>PT</td>
<td>9.51133</td>
</tr>
<tr>
<td>NY</td>
<td>9.43899</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dipeptide</th>
<th>Cumulative Coding Weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>YV</td>
<td>9.43354</td>
</tr>
<tr>
<td>NN</td>
<td>9.39987</td>
</tr>
<tr>
<td>LK</td>
<td>9.39362</td>
</tr>
<tr>
<td>PI</td>
<td>9.3663</td>
</tr>
<tr>
<td>HG</td>
<td>9.23274</td>
</tr>
<tr>
<td>PK</td>
<td>9.22548</td>
</tr>
<tr>
<td>TV</td>
<td>9.20182</td>
</tr>
<tr>
<td>TD</td>
<td>9.12878</td>
</tr>
<tr>
<td>NM</td>
<td>9.11403</td>
</tr>
<tr>
<td>HP</td>
<td>9.10417</td>
</tr>
<tr>
<td>DM</td>
<td>9.08048</td>
</tr>
<tr>
<td>EM</td>
<td>9.04266</td>
</tr>
<tr>
<td>FY</td>
<td>8.98583</td>
</tr>
<tr>
<td>HF</td>
<td>8.97915</td>
</tr>
<tr>
<td>QL</td>
<td>8.97813</td>
</tr>
<tr>
<td>VN</td>
<td>8.92404</td>
</tr>
<tr>
<td>FT</td>
<td>8.89663</td>
</tr>
<tr>
<td>GE</td>
<td>8.86371</td>
</tr>
<tr>
<td>NL</td>
<td>8.80884</td>
</tr>
<tr>
<td>EL</td>
<td>8.80012</td>
</tr>
<tr>
<td>FL</td>
<td>8.7911</td>
</tr>
<tr>
<td>YA</td>
<td>8.77964</td>
</tr>
<tr>
<td>HT</td>
<td>8.77866</td>
</tr>
<tr>
<td>TA</td>
<td>8.77722</td>
</tr>
<tr>
<td>PV</td>
<td>8.77192</td>
</tr>
<tr>
<td>EF</td>
<td>8.77018</td>
</tr>
<tr>
<td>VQ</td>
<td>8.75195</td>
</tr>
<tr>
<td>NI</td>
<td>8.70966</td>
</tr>
<tr>
<td>YK</td>
<td>8.70265</td>
</tr>
<tr>
<td>PM</td>
<td>8.6644</td>
</tr>
<tr>
<td>Dipeptide</td>
<td>Cumulative Coding Weights</td>
</tr>
<tr>
<td>-----------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>DN</td>
<td>8.66317</td>
</tr>
<tr>
<td>WL</td>
<td>8.65014</td>
</tr>
<tr>
<td>FI</td>
<td>8.64432</td>
</tr>
<tr>
<td>YM</td>
<td>8.63627</td>
</tr>
<tr>
<td>ED</td>
<td>8.61801</td>
</tr>
<tr>
<td>HY</td>
<td>8.60462</td>
</tr>
<tr>
<td>NF</td>
<td>8.53631</td>
</tr>
<tr>
<td>PA</td>
<td>8.46067</td>
</tr>
<tr>
<td>QF</td>
<td>8.426</td>
</tr>
<tr>
<td>WF</td>
<td>8.39731</td>
</tr>
<tr>
<td>CE</td>
<td>8.39623</td>
</tr>
<tr>
<td>EA</td>
<td>8.34899</td>
</tr>
<tr>
<td>FM</td>
<td>8.34693</td>
</tr>
<tr>
<td>YE</td>
<td>8.32636</td>
</tr>
<tr>
<td>QQ</td>
<td>8.31699</td>
</tr>
<tr>
<td>NA</td>
<td>8.26595</td>
</tr>
<tr>
<td>VF</td>
<td>8.25225</td>
</tr>
<tr>
<td>QI</td>
<td>8.23292</td>
</tr>
<tr>
<td>GA</td>
<td>8.22043</td>
</tr>
<tr>
<td>EK</td>
<td>8.1995</td>
</tr>
<tr>
<td>QD</td>
<td>8.15482</td>
</tr>
<tr>
<td>TT</td>
<td>8.14748</td>
</tr>
<tr>
<td>PG</td>
<td>8.12627</td>
</tr>
<tr>
<td>DY</td>
<td>8.11778</td>
</tr>
<tr>
<td>EW</td>
<td>8.10057</td>
</tr>
<tr>
<td>GY</td>
<td>8.05289</td>
</tr>
<tr>
<td>DL</td>
<td>8.04066</td>
</tr>
<tr>
<td>HI</td>
<td>7.97765</td>
</tr>
<tr>
<td>TK</td>
<td>7.94104</td>
</tr>
<tr>
<td>QH</td>
<td>7.89374</td>
</tr>
<tr>
<td>NT</td>
<td>7.89274</td>
</tr>
<tr>
<td>PH</td>
<td>7.88197</td>
</tr>
<tr>
<td>HD</td>
<td>7.86328</td>
</tr>
<tr>
<td>HM</td>
<td>7.84305</td>
</tr>
<tr>
<td>DP</td>
<td>7.78763</td>
</tr>
<tr>
<td>QY</td>
<td>7.78197</td>
</tr>
<tr>
<td>LI</td>
<td>7.76398</td>
</tr>
<tr>
<td>VY</td>
<td>7.73824</td>
</tr>
<tr>
<td>HE</td>
<td>7.69111</td>
</tr>
<tr>
<td>YT</td>
<td>7.61348</td>
</tr>
<tr>
<td>GI</td>
<td>7.59054</td>
</tr>
<tr>
<td>GL</td>
<td>7.57192</td>
</tr>
<tr>
<td>TP</td>
<td>7.56612</td>
</tr>
<tr>
<td>SR</td>
<td>7.54091</td>
</tr>
<tr>
<td>FK</td>
<td>7.5001</td>
</tr>
<tr>
<td>WA</td>
<td>7.48522</td>
</tr>
<tr>
<td>TN</td>
<td>7.48029</td>
</tr>
<tr>
<td>HN</td>
<td>7.47486</td>
</tr>
<tr>
<td>NP</td>
<td>7.47394</td>
</tr>
<tr>
<td>WE</td>
<td>7.47226</td>
</tr>
<tr>
<td>CG</td>
<td>7.44065</td>
</tr>
<tr>
<td>FS</td>
<td>7.41148</td>
</tr>
<tr>
<td>HK</td>
<td>7.33513</td>
</tr>
<tr>
<td>WG</td>
<td>7.3044</td>
</tr>
<tr>
<td>QT</td>
<td>7.28106</td>
</tr>
<tr>
<td>DA</td>
<td>7.20384</td>
</tr>
<tr>
<td>GK</td>
<td>7.19007</td>
</tr>
<tr>
<td>TH</td>
<td>7.18526</td>
</tr>
<tr>
<td>PS</td>
<td>7.16827</td>
</tr>
<tr>
<td>WQ</td>
<td>7.16653</td>
</tr>
<tr>
<td>FN</td>
<td>7.15648</td>
</tr>
<tr>
<td>TS</td>
<td>7.11169</td>
</tr>
<tr>
<td>PY</td>
<td>7.08462</td>
</tr>
<tr>
<td>GF</td>
<td>7.05097</td>
</tr>
<tr>
<td>CF</td>
<td>7.04943</td>
</tr>
<tr>
<td>VH</td>
<td>6.98848</td>
</tr>
<tr>
<td>CP</td>
<td>6.96907</td>
</tr>
<tr>
<td>PR</td>
<td>6.94975</td>
</tr>
<tr>
<td>NK</td>
<td>6.94819</td>
</tr>
<tr>
<td>GP</td>
<td>6.94081</td>
</tr>
<tr>
<td>VT</td>
<td>6.92985</td>
</tr>
<tr>
<td>FV</td>
<td>6.91702</td>
</tr>
<tr>
<td>EG</td>
<td>6.90866</td>
</tr>
<tr>
<td>CY</td>
<td>6.90823</td>
</tr>
<tr>
<td>FQ</td>
<td>6.90141</td>
</tr>
<tr>
<td>TF</td>
<td>6.89208</td>
</tr>
<tr>
<td>YF</td>
<td>6.86773</td>
</tr>
<tr>
<td>LM</td>
<td>6.84512</td>
</tr>
<tr>
<td>EN</td>
<td>6.84289</td>
</tr>
<tr>
<td>WY</td>
<td>6.83214</td>
</tr>
<tr>
<td>LT</td>
<td>6.80957</td>
</tr>
<tr>
<td>DG</td>
<td>6.80288</td>
</tr>
<tr>
<td>TI</td>
<td>6.76594</td>
</tr>
<tr>
<td>DS</td>
<td>6.72826</td>
</tr>
<tr>
<td>LQ</td>
<td>6.72267</td>
</tr>
<tr>
<td>MT</td>
<td>6.7058</td>
</tr>
<tr>
<td>ND</td>
<td>6.69236</td>
</tr>
<tr>
<td>YH</td>
<td>6.69028</td>
</tr>
<tr>
<td>RR</td>
<td>6.6801</td>
</tr>
<tr>
<td>EY</td>
<td>6.66046</td>
</tr>
<tr>
<td>CK</td>
<td>6.64021</td>
</tr>
<tr>
<td>CD</td>
<td>6.62721</td>
</tr>
<tr>
<td>WI</td>
<td>6.61583</td>
</tr>
<tr>
<td>EH</td>
<td>6.6156</td>
</tr>
<tr>
<td>HA</td>
<td>6.60928</td>
</tr>
<tr>
<td>DV</td>
<td>6.5538</td>
</tr>
<tr>
<td>PF</td>
<td>6.49424</td>
</tr>
<tr>
<td>CA</td>
<td>6.48236</td>
</tr>
<tr>
<td>LF</td>
<td>6.45457</td>
</tr>
<tr>
<td>TG</td>
<td>6.45109</td>
</tr>
<tr>
<td>GH</td>
<td>6.43565</td>
</tr>
</tbody>
</table>

106
<table>
<thead>
<tr>
<th>Dipeptide</th>
<th>Cumulative Coding Weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>WN</td>
<td>6.43231</td>
</tr>
<tr>
<td>EV</td>
<td>6.42914</td>
</tr>
<tr>
<td>DW</td>
<td>6.3698</td>
</tr>
<tr>
<td>HH</td>
<td>6.36351</td>
</tr>
<tr>
<td>DQ</td>
<td>6.34581</td>
</tr>
<tr>
<td>HS</td>
<td>6.33577</td>
</tr>
<tr>
<td>NH</td>
<td>6.33349</td>
</tr>
<tr>
<td>VP</td>
<td>6.32453</td>
</tr>
<tr>
<td>QK</td>
<td>6.31186</td>
</tr>
<tr>
<td>CN</td>
<td>6.30609</td>
</tr>
<tr>
<td>YI</td>
<td>6.27458</td>
</tr>
<tr>
<td>WD</td>
<td>6.25846</td>
</tr>
<tr>
<td>GQ</td>
<td>6.24454</td>
</tr>
<tr>
<td>CH</td>
<td>6.1787</td>
</tr>
<tr>
<td>WM</td>
<td>6.1572</td>
</tr>
<tr>
<td>NQ</td>
<td>6.14934</td>
</tr>
<tr>
<td>MQ</td>
<td>6.12851</td>
</tr>
<tr>
<td>WT</td>
<td>6.10647</td>
</tr>
<tr>
<td>QN</td>
<td>6.07118</td>
</tr>
<tr>
<td>NR</td>
<td>6.05014</td>
</tr>
<tr>
<td>GT</td>
<td>6.04584</td>
</tr>
<tr>
<td>DK</td>
<td>6.04103</td>
</tr>
<tr>
<td>LH</td>
<td>6.00383</td>
</tr>
<tr>
<td>TY</td>
<td>5.99062</td>
</tr>
<tr>
<td>LP</td>
<td>5.97024</td>
</tr>
<tr>
<td>TR</td>
<td>5.96029</td>
</tr>
<tr>
<td>GS</td>
<td>5.95335</td>
</tr>
<tr>
<td>CV</td>
<td>5.89957</td>
</tr>
<tr>
<td>NW</td>
<td>5.8192</td>
</tr>
<tr>
<td>VW</td>
<td>5.81078</td>
</tr>
<tr>
<td>MK</td>
<td>5.80918</td>
</tr>
<tr>
<td>WK</td>
<td>5.76737</td>
</tr>
<tr>
<td>PP</td>
<td>5.7531</td>
</tr>
<tr>
<td>AR</td>
<td>5.71423</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dipeptide</th>
<th>Cumulative Coding Weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>LR</td>
<td>5.69692</td>
</tr>
<tr>
<td>VR</td>
<td>5.66414</td>
</tr>
<tr>
<td>VI</td>
<td>5.62658</td>
</tr>
<tr>
<td>WH</td>
<td>5.6212</td>
</tr>
<tr>
<td>YG</td>
<td>5.61592</td>
</tr>
<tr>
<td>CS</td>
<td>5.59296</td>
</tr>
<tr>
<td>VS</td>
<td>5.58522</td>
</tr>
<tr>
<td>YP</td>
<td>5.51026</td>
</tr>
<tr>
<td>QV</td>
<td>5.50813</td>
</tr>
<tr>
<td>CR</td>
<td>5.50163</td>
</tr>
<tr>
<td>QR</td>
<td>5.49508</td>
</tr>
<tr>
<td>TW</td>
<td>5.44798</td>
</tr>
<tr>
<td>ML</td>
<td>5.4217</td>
</tr>
<tr>
<td>KS</td>
<td>5.39961</td>
</tr>
<tr>
<td>QS</td>
<td>5.39269</td>
</tr>
<tr>
<td>MY</td>
<td>5.39181</td>
</tr>
<tr>
<td>MM</td>
<td>5.383</td>
</tr>
<tr>
<td>DF</td>
<td>5.37065</td>
</tr>
<tr>
<td>GN</td>
<td>5.36784</td>
</tr>
<tr>
<td>PW</td>
<td>5.35799</td>
</tr>
<tr>
<td>LS</td>
<td>5.34033</td>
</tr>
<tr>
<td>LY</td>
<td>5.315</td>
</tr>
<tr>
<td>WS</td>
<td>5.2734</td>
</tr>
<tr>
<td>CW</td>
<td>5.25911</td>
</tr>
<tr>
<td>CT</td>
<td>5.22478</td>
</tr>
<tr>
<td>FW</td>
<td>5.19295</td>
</tr>
<tr>
<td>FF</td>
<td>5.18668</td>
</tr>
<tr>
<td>MR</td>
<td>5.18137</td>
</tr>
<tr>
<td>CQ</td>
<td>5.16586</td>
</tr>
<tr>
<td>GR</td>
<td>5.14588</td>
</tr>
<tr>
<td>MN</td>
<td>5.14347</td>
</tr>
<tr>
<td>MI</td>
<td>5.12758</td>
</tr>
<tr>
<td>TC</td>
<td>5.06582</td>
</tr>
<tr>
<td>HQ</td>
<td>5.04734</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dipeptide</th>
<th>Cumulative Coding Weights</th>
</tr>
</thead>
<tbody>
<tr>
<td>EP</td>
<td>4.9811</td>
</tr>
<tr>
<td>YN</td>
<td>4.98046</td>
</tr>
<tr>
<td>HW</td>
<td>4.95606</td>
</tr>
<tr>
<td>EI</td>
<td>4.91192</td>
</tr>
<tr>
<td>LW</td>
<td>4.91087</td>
</tr>
<tr>
<td>MS</td>
<td>4.71925</td>
</tr>
<tr>
<td>FH</td>
<td>4.71811</td>
</tr>
<tr>
<td>FP</td>
<td>4.66208</td>
</tr>
<tr>
<td>CI</td>
<td>4.58142</td>
</tr>
<tr>
<td>CC</td>
<td>4.52851</td>
</tr>
<tr>
<td>ET</td>
<td>4.48002</td>
</tr>
<tr>
<td>QW</td>
<td>4.47265</td>
</tr>
<tr>
<td>HR</td>
<td>4.29306</td>
</tr>
<tr>
<td>DR</td>
<td>4.2894</td>
</tr>
<tr>
<td>DT</td>
<td>4.14813</td>
</tr>
<tr>
<td>FC</td>
<td>4.1186</td>
</tr>
<tr>
<td>ER</td>
<td>4.11122</td>
</tr>
<tr>
<td>GC</td>
<td>4.07452</td>
</tr>
<tr>
<td>FR</td>
<td>4.06965</td>
</tr>
<tr>
<td>MC</td>
<td>4.05298</td>
</tr>
<tr>
<td>YW</td>
<td>4.01986</td>
</tr>
<tr>
<td>DH</td>
<td>3.98926</td>
</tr>
<tr>
<td>EC</td>
<td>3.94789</td>
</tr>
<tr>
<td>MF</td>
<td>3.94705</td>
</tr>
<tr>
<td>MA</td>
<td>3.86243</td>
</tr>
<tr>
<td>YY</td>
<td>3.83317</td>
</tr>
<tr>
<td>MW</td>
<td>3.82562</td>
</tr>
<tr>
<td>NC</td>
<td>3.75993</td>
</tr>
<tr>
<td>ES</td>
<td>3.7225</td>
</tr>
<tr>
<td>LC</td>
<td>3.7171</td>
</tr>
<tr>
<td>VC</td>
<td>3.64949</td>
</tr>
<tr>
<td>WR</td>
<td>3.63512</td>
</tr>
<tr>
<td>QP</td>
<td>3.58443</td>
</tr>
<tr>
<td>WP</td>
<td>3.50352</td>
</tr>
<tr>
<td>Dipeptide</td>
<td>Cumulative Coding Weights</td>
</tr>
<tr>
<td>-----------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>PC</td>
<td>3.43567</td>
</tr>
<tr>
<td>MP</td>
<td>3.33801</td>
</tr>
<tr>
<td>YC</td>
<td>3.2306</td>
</tr>
<tr>
<td>MV</td>
<td>3.19438</td>
</tr>
<tr>
<td>KW</td>
<td>3.19228</td>
</tr>
<tr>
<td>YS</td>
<td>3.18682</td>
</tr>
<tr>
<td>NS</td>
<td>3.15582</td>
</tr>
<tr>
<td>WW</td>
<td>3.10391</td>
</tr>
<tr>
<td>QC</td>
<td>2.98028</td>
</tr>
<tr>
<td>HC</td>
<td>2.92722</td>
</tr>
<tr>
<td>YR</td>
<td>2.68459</td>
</tr>
<tr>
<td>DC</td>
<td>2.52902</td>
</tr>
<tr>
<td>WC</td>
<td>2.35417</td>
</tr>
<tr>
<td>CM</td>
<td>2.04526</td>
</tr>
<tr>
<td>MH</td>
<td>1.94291</td>
</tr>
<tr>
<td>CL</td>
<td>0</td>
</tr>
<tr>
<td>HV</td>
<td>0</td>
</tr>
<tr>
<td>LA</td>
<td>0</td>
</tr>
</tbody>
</table>