Abstract

MAY, ELEBEOBA ENI. Analysis of Coding Theory Based Models for Initiating Protein Translation in Prokaryotic Organisms *(Under the direction of Dr. Mladen Vouk and Dr. Donald Bitzer).*

Rapid advances in both genomic data acquisition and computational technology have encouraged the development and use of engineering methods in the field of bioinformatics and computational genomics. Several researchers are encouraging the use of error-correction coding in analyzing genetic data [1, 2]. A goal of current work in this context is to use coding theory analysis to determine whether regions of the specified genome are protein-producing sequences.

Using information theory, coding theory specifically, this work develops a coding theory view of the translation initiation process in prokaryotic organisms. The translation of messenger RNA into amino acid sequences is functionally paralleled to the decoding of noisy, convolutionally encoded parity streams. This work presents a genetic algorithms method for the design of optimal table-based convolutional coding models for translation initiation sites using *Escherichia coli* K-12 as the model organism. Sequence and function-based convolutional coding models are constructed and applied to prokaryotic organisms of varying taxonomical relation including: *Escherichia coli* K-12, *Salmonella typhimurium* LT2, *Bacillus subtilis*, and *Staphylococcus aureus* Mu50. Several categories of error-control codes are explored and compared, including: horizontal versus vertical codes and equal versus unequal error protection (UEP) codes.

This work produced convolutional codes with decoding masks having high similarity to the 3' end of the 16S ribosomal RNA. Results show that UEP code models recognize the non-random and Shine-Dalgarno domain better than equal error protection models. But, equal error protection models are more effective error detectors. Testing indicates that function-based models are more likely to distinguish taxonomical differences than sequence-based models. Additional results are presented.

Research contributions include: coding theory view of prokaryotic translation initiation, the first table-based, convolutional coding model development and design methodology for prokaryotic translation initiation, the first set of (3,1,4) error-control coding models for translation initiation, comparative analysis of models on prokaryotic organisms of differing taxonomical relatedness, and extension of table-based coding principles to field five.

Analysis of Coding Theory Based Models for Initiating Protein Translation in Prokaryotic Organisms

by
Elebeoba Eni May

A thesis submitted to the Graduate Faculty of

North Carolina State University

in partial fulfillment of the

requirements for the Degree of
Doctor of Philosophy

**Department of
Electrical and Computer Engineering**

Raleigh, NC

March 29, 2002

Approved By:

Co-Chairman of Advisory Committe

Chairman of Advisory Committee

TO MY FATHER UDE EKE ENI, M.D.

AND

MY MOTHER GLORY UDE ENI

Biography

Elebeoba Eni May was born in Abiriba, Abia (formerly Imo) State, Nigeria on May 27, 1973 to Dr. Ude Eke Eni and Mrs. Glory Ude Eni. She is the third of four children. Elebeoba spent most of her childhood years in Durham, North Carolina. She is a 1991 graduate of the North Carolina School of Science and Mathematics, a residential high school for academically gifted students in North Carolina. During her senior year she received several academic scholarships, including the Dupont Minority Scholarship and the prestigious John T. Caldwell Scholarship from North Carolina State University (NCSU).

While pursuing her bachelor degree in Computer Engineering at North Carolina State University, Elebeoba participated in the Benjamin Franklin Scholars Program and the NCSU University Fellows Program. Towards the end of her undergraduate tenure at NCSU she began working as an undergraduate research assistant under the direction of Dr. Winser Alexander.

In 1996, Elebeoba graduated with honors from NCSU, where she continued her studies as a graduate student in the Electrical and Computer Engineering Department under the direction of Dr. Mladen Vouk and Dr. Donald Bitzer. Elebeoba May received her masters of science degree in May of 1999 and her doctorate degree in May of 2002. During her graduate tenure at North Carolina State University she received several prestigious fellowships and awards including, the National Science Foundation Minority Graduate Fellowship, the Ford Foundation Dissertation Fellowship for Minorities, the Association for the Concerns of African American Graduate Student Graduate Research Award, and was inducted into the Eta Kappa Nu Honor Society.

Acknowledgements

"It is the glory of God to conceal a matter; to search out a matter is the glory of kings."
Proverbs 25:2 NKJV

First and foremost I thank my Lord, and my Savior Jesus Christ for doing "exceedingly abundantly above all that [I] was able to ask or think." I thank my Heavenly Father for the hidden mysteries of His creation and for the opportunity to search out the intricate beauty of His world.

I would like to thank my advisors, Dr. Mladen Vouk and Dr. Donald Bitzer for their guidance, encouragement, and support in this research. I also would like to extend my gratitude to Dr. Anne Stomp, Dr. Jeff Thorne, and Dr. Winser Alexander for being part of my committee and for their insight and interest in my work. I would especially like to thank Dr. Winser Alexander for his continued support throughout my graduate tenure.

To my husband, Mr. Kevin J. May, thank you for being absolutely wonderful. Thank you for the sacrifices you made so that I could reach for my dream. You gave me confidence and hope when this road became an uphill climb. Your are the joy of my life; you put the twinkle in my eyes. For us forever is only the beginning.

I continue to be extremely grateful for and to my family who have been my greatest support. For this I thank each of them: my parents Dr. Ude and Mrs. Glory Eni, my sister Mrs. Ubani Asiegbu, and my brothers Pastor Eke Eni and Mr. Egbe Eni. You have always been in my corner, encouraging me to persevere. I did not travel this road alone. This accomplishment is not mine alone. Thank you for sharing in my struggles and my victories.

Thank you to my friends and colleagues for their support, suggestions, and continued encouragement; thank you to: Dr. Sonetra Howard, Dr. Tamara Baynham, Ms. Dedra Eatmon, Ms. Tiffany Barnes, and Dr. David Rosnick.

This research is supported in part by a National Science Foundation Minority Graduate Fellowship, the Ford Foundation Dissertation Fellowship for Minorities, and a National Science Foundation MGE Grant.

Table of Contents

List of Figures x

List of Tables xiii

1 Introduction 1

1.1 Problem Statement 1

1.1.1 Research Objectives 3

1.1.2 Research Contributions 5

1.2 Review of Gene Identification Algorithms and Methods 6

1.2.1 Template Methods 6

1.2.2 Similarity Searches 10

1.3 Information Theory and Gene Identification 12

1.3.1 Information Theory Framework for Biological Information Processing 13

1.3.2 Information Based Models 15

1.4 Coding Theory and the Gene Identification Problem 21

1.4.1 Coding Theory Based Models 24

1.4.2 Coding Theory and DNA Computing [3] 24

1.5 Document Organization 26

2 Theoretical Background 27

2.1 Theoretical Background 28

2.2 Previous Code Models for Translation Initiation[4][5][6] 30

2.2.1 Error Control Coding Based Methods 31

2.2.2 Block Code Based Maximum-Likelihood Classifier 36

3 Table-Based Coding Systems 40

3.1 Basics of Convolutional Coding 41

3.1.1 Encoding Methodology 41

3.1.2 Decoding Methodology 43

3.2 Table-Based Codes 43

3.2.1 Table-Based Encoding 43

3.2.2 Table-Based Decoding 47

3.2.3 Formation of the Gmask 49
3.3 Table-Based Codes for Field of Five 50
3.3.1 Base Five Table-Based Encoding 53
3.3.2 Base Five Table-Based Decoding 56
3.3.3 Formation of the Gmask 57

4 Table-Based Codes for Prokaryotic Translation Initiation Systems 64
4.1 Finding Optimal Convolutional Codes 64
4.2 Genetic Algorithms [7] 65
4.2.1 Overview of Genetic Algorithms 65
4.2.2 Components of a Genetic Algorithm 67
4.2.3 Genetic Operators 69
4.3 Genetic Algorithms for Table-Based Translation Initiation Codes 76
4.3.1 Methodology 78
4.3.2 Procedure 95
4.3.3 Results 99
4.3.4 Discussion 103

5 Constructing Codes from Binding Motifs 105
5.1 Binding Analysis of Translation Initiation Sites 106
5.1.1 Functional Definition of mRNA Leader and Ribosomal Interaction . 106
5.1.2 Binding Pattern Representation 108
5.1.3 Binding Analysis Applied to Leader and Non-leader mRNA Sequences 108
5.1.4 From Binding Vectors to Codewords 112
5.2 Genetic Algorithms for Binding-Based Translation Initiation Codes 113
5.2.1 Methodology 113
5.2.2 Procedure 123
5.2.3 Results 127
5.2.4 Discussion 129

6 Evaluation of Coding Models for Prokaryotic Translation Initiation 131
6.1 Base Five Table-Based Code Models 133
6.1.1 *E. coli* K-12 133
6.1.2 Application to Other Prokaryotic Organisms: *S. typhimurium* LT2, *B. subtilis*, and *S. aureus* Mu50 139
6.1.3 Discussion 146
6.2 Binary Table-Based Convolutional Code Models 147
6.2.1 *E. coli* K-12 148
6.2.2 Application to Other Prokaryotic Organisms: *S. typhimurium* LT2, *B. subtilis*, and *S. aureus* Mu50 156
6.2.3 Discussion 163

7 Summary and Conclusion **165**
7.1 Analysis of Table-Based Convolutional Coding Models 166
7.1.1 Code Model Versus Biological Models [8] 166
7.1.2 Analysis of Models 167
7.2 Research Implications and Contributions 169
7.3 Future Research 171

List of References **174**

List of Figures

1.1 Information Theory Based View of Protein Synthesis (Roman-Roldan, et al.) 13
1.2 Central Dogma of Genetics as a Coding System (May, et al.) 14
1.3 Modified Coding Theory View of the Central Dogma of Genetics 23

2.1 Communication View of the Central Dogma of Genetics 30
2.2 Results of Minimum Distance Block Decoding Model for (5,2) Code 32
2.3 Frequency of Two-Pattern Syndrome Distance Values 34
2.4 Probability Distributions for (5,2) Maximum-Likelihood Classifiers 39

3.1 A (2,1,2) Convolutional Encoder 42

4.1 Average Syndrome Distance for Base Five Table-Based Convolutional Code Models for Translation Initiation 100
4.2 Average Syndrome Distance From the All-Zero Syndrome for Base Five Table-Based Convolutional Code Models - Vertical Models 100
4.3 Average Similarity Between Exposed Part of the 16S rRNA and the Gmasks of Base Five Table-Based Convolutional Code Models for Translation Initiation 102
4.4 Individual Fitness versus Individual Similarity Values for Base Five Code Models 103

5.1 Percent of Sequences with Binding Pattern of 4,0,0 and Above 109
5.2 Positional Binding Ratio of Translated Sequence Group to Non-Translated Sequence Group 111
5.3 Average Syndrome Distance From the All-Zero Syndrome for Binary Table-Based Convolutional Code Models for Translation Initiation 127
5.4 Average Gmasks Values for Binary Table-Based Convolutional Code Models for Translation Initiation 129

6.1 Average Syndrome Distance for *E. coli* Test Set Tested with Base Five Code Models 134
6.2 Fitness Distribution for *E. coli* Test Set and Base Five Code Models 135
6.3 Base Five Code Usage Ratio for *E. coli* Test Set 136

6.4 Average Syndrome Distance for *E. coli* Test Set Tested with Base Five Motif-Based Code Models . . . 137
6.5 Fitness Distribution for *E. coli* Test Set and Base Five Motif-Based Code Models . . . 138
6.6 Base Five Motif-Based Code Usage Ratio for *E. coli* Test Set . . . 139
6.7 Average Syndrome Distance for Prokaryotic Test Set Tested with Base Five Code Models . . . 140
6.8 Fitness Distribution for Prokaryotic Test Set and Base Five Code Models . 141
6.9 Base Five Code Usage Ratio for *B. subtilis* Test Set . . . 141
6.10 Base Five Code Usage Ratio for *S. typhimurium* Test Set . . . 142
6.11 Base Five Code Usage Ratio for *S. aureus* Test Set . . . 142
6.12 Average Syndrome Distance for Prokaryotic Test Set Tested with Motif-Based Base Five Code Models . . . 143
6.13 Fitness Distribution for Prokaryotic Test Set and Motif-Based Base Five Code Models . . . 144
6.14 Motif-Based Base Five Code Usage Ratio for *B. subtilis* Test Set . . . 145
6.15 Motif-Based Base Five Code Usage Ratio for *S. typhimurium* Test Set . . . 145
6.16 Motif-Based Base Five Code Usage Ratio for *S. aureus* Test Set . . . 146
6.17 Average Syndrome Distance for *E. coli* Test Set Tested with Binary Code Models . . . 148
6.18 Average Syndrome Distance for *E. coli* Test Set Tested with Binary Code Models - Penalty for Zero . . . 149
6.19 Fitness Distribution for *E. coli* Test Set and Binary Code Models . . . 150
6.20 Fitness Distribution for *E. coli* Test Set and Binary Code Models - Penalty for Zero . . . 150
6.21 Binary Code Usage Ratio for *E. coli* Test Set . . . 151
6.22 Binary Code Usage Ratio for *E. coli* Test Set - Penalty for Zero . . . 152
6.23 Average Syndrome Distance for *E. coli* Test Set Tested with Binary Motif-Based Code Models . . . 152
6.24 Average Syndrome Distance for *E. coli* Test Set Tested with Binary Motif-Based Code Models - Penalty for Zero . . . 153
6.25 Fitness Distribution for *E. coli* Test Set and Binary Motif-Based Code Models 154
6.26 Fitness Distribution for *E. coli* Test Set and Binary Motif-Based Code Models - Penalty for Zero . . . 154
6.27 Binary Motif-Based Code Usage Ratio for *E. coli* Test Set . . . 155
6.28 Binary Motif-Based Code Usage Ratio for *E. coli* Test Set - Penalty for Zero 156
6.29 Average Syndrome Distance for Prokaryotic Test Set Tested with Binary Code Models . . . 157
6.30 Average Syndrome Distance (Position -20 to -9) for Prokaryotic Test Set Tested with Binary Code Models . . . 158
6.31 Fitness Distribution for Prokaryotic Test Set and Binary Code Models . . 158
6.32 Binary Code Usage Ratio for *B. subtilis* Test Set . . . 159

6.33 Binary Code Usage Ratio for *S. typhimurium* Test Set 159
6.34 Binary Code Usage Ratio for *S. aureus* Test Set 160
6.35 Average Syndrome Distance for Prokaryotic Test Set Tested with Motif-Based Binary Code Models 161
6.36 Fitness Distribution for Prokaryotic Test Set and Motif-Based Binary Code Models 161
6.37 Motif-Based Binary Code Usage Ratio for *B. subtilis* Test Set 162
6.38 Motif-Based Binary Code Usage Ratio for *S. typhimurium* Test Set 162
6.39 Motif-Based Binary Code Usage Ratio for *S. aureus* Test Set 163

List of Tables

1.1 Performance of Markov-based Gene Identification Methods. 9

2.1 Results of M-L Classifier for (5,2) Block Code 38
2.2 Classification Rates for (5,2) M-L Classification Systems 38

3.1 Reduced Encoding Table 46
3.2 Reduced Decoding Table 48
3.3 Field 5 multiplication operation in relation to RNA bases. 51
3.4 Field 5 addition operation in relation to RNA bases. 51
3.5 Field 5 division operation in relation to RNA bases. 51
3.6 Field 5 subtraction operation in relation to RNA bases. 52
3.7 Reduced Encoding Table for (3,1,4) Base 5 Convolutional Code 55
3.8 Reduced Decoding Table for (3,1,4) Base 5 Convolutional Code 56

4.1 Mapping of RNA bases to the finite field of five. 76
4.2 Conversion of Binary Chromosome to Application Specific Parameters . . . 82
4.3 Regional weighting distribution for motif-based fitness calculation 93
4.4 Motif weights by position on mRNA leader 93
4.5 Contents of summary data file 97
4.6 Contents of the all log file 98

5.1 Distribution of Strongest Binding Patterns for Translated, Non-translated, and Random Sequence Groups 110
5.2 Location of Largest Translated to Non-translated Positional Binding Ratio Value 112
5.3 Regional weighting distribution for motif-based fitness calculation 122
5.4 Contents of summary data file 125
5.5 Contents of the all log file 126

6.1 Contents of test data sets' summary and all data file 132
6.2 Relationship between number of zeros in syndrome vector and fitness value 135

Chapter 1

Introduction

The central premise of genetics is that genetic information is perpetuated in the form of nucleic acid sequences but functions once expressed as proteins [9]. Various investigators have developed models that attempt to capture different information related aspects of the genetic system. Most of these models are based on the information contained in DNA sequences. Since the advent of diverse genome projects, the amount of sequence data has grown exponentially. With the increased availability of genomic data, the research emphasis has shifted from sequence compilation techniques to genomic sequence and system analysis.

1.1 Problem Statement

Redundancy occurs within genomic sequences [9]. The DNA, which exists in a double helix form, is itself doubly redundant. Hence, at the chromosomal level, we can consider unreplicated DNA as a half-rate code. At the nucleic acid level, messenger RNA (mRNA) sequences are mapped to amino-acid bases by grouping three nucleic acid bases together to form a codon. A codon, or three base nucleic acid vector, is mapped to a single amino acid information. This process which occurs during translation can be viewed as the decoding of a one third-rate code. There are possibly many levels of error control coding throughout genetic sequences and systems, including the systems which govern protein folding, structure and function. Although this work focuses on single dimensional codes, the genetic codes most likely exist in higher-dimension coding spheres. This work investigates coding in the traditional sense, or horizontal codes, and coding in the vertical direction. Horizontal codes correspond to a coding model in which a single encoder produces the encoded genetic sequence for a single receiver. The vertical coding model corresponds to a multiple receiver model, where the message is encoded such that specific regions within the sequence are recognized by specific receivers.

Research in information based sequence analysis has shown that ribosome binding sites evolve to functional requirements rather than perfect sequences [10]. Given the number of times that genetic information is transmitted and correctly interpreted and the size of an organism's genome, the genetic system must employ error-correction methods in order for genetic information, and ultimately an organism, to survive [1].

Previous work (summarized in Section 2.2) [11][4][5][6] suggests that it may be possible to design an error-control coding based algorithm for detecting the leader regions of translated messenger RNA sequences in prokaryotic genomes. The results of the block code, maximum-likelihood classifier [6] validate the plausibility of designing and implementing an error-control coding based method that determines whether an individual mRNA sequence can successfully initiate translation. The relative success of the sliding block-code model combined with preliminary results from the convolutional coding model leads to the belief that the ribosome contains memory and can more accurately be paralleled to a convolutional decoder. Based on prior results, the following working assumptions are formed:

- The genetic replication-transcription-translation process can be paralleled to an engineering communication system as depicted in Figure 2.1.
- The base five mapping of RNA bases to numeric values between zero and four is a plausible mapping.

- The 3' end of the 16S rRNA is involved in the decoding of the leader region. Therefore, the genetic sequence of the 3' end of the 16S rRNA can be used to evaluate, determine and/or develop decoding masks for prokaryotic mRNA sequences.
- The decision to initiate translation depends on the cumulative information of bases at least twenty positions upstream of the initiation codon.
- The mRNA sequence can be modeled as a convolutionally encoded parity sequence and the ribosome can be described as a convolutional decoder for the mRNA sequence.

1.1.1 Research Objectives

The principle hypothesis governing this work is, given that redundancy naturally occurs within nucleic acid sequences and the genetic sequence is mutated by error generating processes such as replication, it is plausible to view messenger RNA (mRNA) as a noisy encoded signal and the ribosome as a decoder which uses memory to recognize valid initiation sites. If the redundancy, or extra information, contained in the genetic sequence is used to protect the organism against errors then it would be feasible to use principles of error control coding theory to interpret the genetic translation initiation mechanism. If the hypothesis is realizable then there may exist a valid set of table-based, convolutional, error-control codes that: 1) Have decoding masks (gmasks) that are similar to the 3' end of the 16S rRNA; 2) Have decoding masks which identify key sites on the mRNA leader (5' untranslated) region that are involved in translation initiation; 3) Have decoding masks that can be used to detect valid and invalid initiation sites.

Previous work (discussed in Chapter 2) and related literature support the coding theory view of translation initiation and other genetic processes. Having established the plausibility of the coding theory model for *E. coli* K-12 in the general sense, the goal of current work is to develop an error-control coding technique for designing and evaluating convolutional code models for prokaryotic translation initiation systems in the specific sense. The goal is realized through the following objectives:

1. In depth analysis of the translation initiation system using table-based convolutional coding techniques, specifically table-based encoding and syndrome-based decoding methods.

2. Design of $(3, 1, 4)$ convolutional coding models for translation initiation using *E. coli* K-12 as model organism. The $(3, 1, 4)$ code is used to illustrate and test the table-based, code construction methodology developed in this work.

   - Design of sequence based convolutional code models.
   - Design of function based convolutional code models.

3. Application and analysis of convolutional code models to other prokaryotic organisms. Research issues to investigate include:

   - Error detecting ability of convolutional code models.
   - Models' response and sensitivity to prokaryotic organisms that are closely related or distant relatives (in a taxonomical sense) of *E. coli* K-12.

   Prokaryotic organisms investigated include:

   - *Salmonella typhimurium* LT2
   - *Bacillus subtilis*
   - *Staphylococcus aureus* Mu50

1.1.2 Research Contributions

The need for error-control coding models has been suggested by researchers such as Battail [1], Schneider [10] and Eigen [12]. Currently, there are few researchers investigating error-control coding methods for analyzing genetic systems and there are no known coding theory models or classification systems, besides our previous block code system, for prokaryotic translation initiation. Although researchers, including L. Kari [3] a faculty in the Computer Science Department at the University Ontario, are beginning to investigate coding theory models for genetic processes, this field is still in its infancy.

This work contributes to the field of computational bioinformatics and biology through the application of information theory, communication theory and coding theory principles to the study and analysis of prokaryotic translation initiation. Specific contributions of this work include:

- Development of a coding theory view of the translation initiation process in prokaryotic organisms based on the parallelism between decoding of parity streams and the translation of mRNA into amino acid sequences. The coding model proposed by this work is based on a nested encoding and decoding model.
- Exploration and implementation of the first known table-based, convolutional-coding model development and design techniques for prokaryotic translation initiation systems using syndrome-based methods. The methods investigated in this work can be used to construct code models which describe individual mRNA leader sequences and the functional behavior of the translation initiation system.
- The first set of (3,1,4) error-control coding based models for prokaryotic translation initiation.
- Testing system and tools for performing comparative analysis of coding models for prokaryotic organisms of differing taxonomical relatedness.

- Extension of table-based coding principles to field five convolutional codes.

1.2 Review of Gene Identification Algorithms and Methods

Fickett [13] defines the gene identification dilemma (from the developers standpoint) as the problem of using computers to decipher nucleotide sequences for the purpose of locating and defining the structure and the functionality of protein producing genes. The goal of gene identification is the automatic annotation of genomic sequences. This means, given a nucleotide sequence, the identification system can determine all regions that are biochemicaly active and describe the reaction that occurs and the products of said reaction [13]. The long-range goals of gene identification include the development of algorithms for the purpose of designing new genes and genomes [13].

Computational techniques used in developing gene identification systems can be categorized as either template methods or lookup methods. Template methods use prototypes to identify genes. Unknown genes are compared to a prototype and classified based on a pre-defined metric [13]. The lookup method, or similarity search, compares the unknown gene or nucleotide sequence to known genes or gene components stored in databases [13]. While the template method adds to our understanding of the gene, it excludes outliers or important exceptions to the gene prototype [13]. Although the lookup method may include exceptions, sequencing errors can result in serious problems when using a look up method [13].

1.2.1 Template Methods

The majority of computational gene identification techniques reviewed in this section can be classified as template methods. Although some methods do not always fit neatly into a specific category, the following four categories will be used to classify computational techniques discussed:

- Hidden Markov Methods
- Signal Processing Methods
- Machine Learning Methods
- Information Theory Methods

The basis of many gene identification systems is a value Fickett refers to as a coding measure. For a given sequence window, a coding measure assigns to a sequence a value or vector of values which correspond to the probability that the sequence is a protein coding sequence [13]. Some systems employ various combinations of computational methods and coding measures for the purpose of gene identification and classification. Fickett found that combining several measures improves the accuracy of the identification system [13].

Hidden Markov Methods

Many widely used gene identification tools are based on Hidden Markov Models (HMM). Hidden Markov Models are used to model systems with hidden discrete states [14]. Biological sequences, such as nucleic acid sequences, can be modeled as the output of a process that goes through discrete states, v. The modeling assumption, or Markov assumption, on which HMMs are based, is that the states that follow any state, v, depend only on v and are independent of all states which precede v [14]. The HMM is defined by a set of possible states and transitions. Each state is associated with a discrete output probability distribution. There are also transition probabilities for each possible transition from a given state. The sum of all transition probabilities must be equal to one for a given state, v [14].

HMMs, which are also used in speech recognition, have been applied to various biological systems [15, 16, 17, 18, 14, 19]. Once the HMM is developed for a particular sequence or group of sequences, it is then used to classify sequences whose functions are not yet determined. For example, the HMM modeling approach was used by Henderson, Salzberg, and Fasman [14] to develop VEIL (Viterbi Exon-Intron Locator), an HMM system that segments uncharacterized eukaryotic DNA sequences into exons, introns, and intergenic regions. This HMM uses the Viterbi decoding algorithm, which is also used in coding theory [20], to determine the probability of the HMM generating the observed sequence.

Another HMM based model, HMMER, was designed by Eddy [19]. HMMER provides tools for constructing an HMM model from initially unaligned training nucleotide sequences. This software provides a useful tool for modeling sequences that may have similar characteristics. Once the model is constructed, it can then be used to determine how well uncharacterized sequences align with the sequences the HMM models.

The HMM method has also been used to develop other gene identification tools such as GeneMARK.hmm [15] and Meta-MEME (a motif-based HMM for protein sequences) [18]. GeneMark.hmm is derived from GENEMARK [21]. GENEMARK uses a Markov chain model in its gene identification algorithm [15]. Krogh, Mian, and Haussler [17] use a hidden Markov model to locate genes in *E. coli* nucleotide sequences.

Table 1.1 summarizes the reported performance values for three Markov based gene identification methods [14, 17, 21]. In Table 1.1, "sensitivity" represents the percent of whole exons that are predicted exactly and "specificity" indicates the percent of exons predicted that are exactly correct [14]. The term "non-homologous" refers to test sequences that are less than eighty percent identical to training set sequences [14]. The GENEMARK results are for a control set of ninety-six base pair fragments of *E. coli* DNA [21].

Table 1.1: Performance of Markov-based Gene Identification Methods.

HMM Method	Organism	Reported Performance
VEIL	Eukaryotes	*Sensitivity:* Homologous - 53% Non-homologous - 50%
		Specificity: Homologous - 49% Non-homologous - 47%
Krogh et al.(1994)	Prokaryote (*E. coli*)	80% of 240 test set genes.
GENEMARK (1993)	Prokaryote (*E. coli*)	False Negative - 10.0% False Positive - 25.2%

Signal Processing Based Methods

Coding measures derived from signal processing constructs such as Fourier Transforms [22] and Wavelet Transforms [23] have been used in developing gene recognition algorithms and analyzing genetic sequences.

Veljkovic [24] and Cosic [22] used Fourier analysis of DNA, RNA, and protein sequences to find a parameter that relates the biological function of nucleic acid and amino acid sequences to their functionality. Using cross-spectral and spectral analysis, the biological signal was analyzed as a finite-length deterministic signal. Since there are examples of biologically unrelated sequences that have great homology, Veljkovic and Cosic's work introduced a new approach for functionally evaluating and classifying biological sequences. They evaluated sequences based on their frequency domain characteristics. Cosic used spectral analysis as the foundation for her resonant recognition model (RRM) which showed a correlation between the biological function of a sequence and the frequencies which are present within the biological signal.

Arneodo et al. [23] developed the Wavelet Transform Modulus Maxima (WTMM) method and applied it to human genomic sequences. Arneodo et al. claim that WTMM analysis can provide a definite answer to questions regarding the long-range correlation properties of DNA coding and non-coding sequences. The wavelet transform, which is the basis of the WTMM method, is able to characterize the scaling properties of fractal objects or signals even in the presence of low frequency trends. Arneodo et al. conclude that introns or non-coding subsequences behave as positively correlated fraction Brownian Motion while coding regions, exons, behave like uncorrelated ordinary Brownian motion [23]. Results of this Wavelet Transform based analysis of nucleic acid sequences could be used in designing gene identification algorithms that classify sequences as protein- coding or non-coding.

Machine Learning Based Methods

Machine learning techniques such as neural networks are also used in the analysis and classification of nucleic acid sequences. For example, a multiple sensor neural network was used by Uberbacher and Mural [25] to elucidate protein-coding regions in human DNA sequences. They used seven algorithms whose results serve as inputs to the neural network. They evaluate a ninety-nine base window. The seven values indicate the likelihood that a given sequence position is part of a coding region. After training the neural network and extracting the weights, the neural network is used to characterize human DNA sequences as coding or non-coding. Neural network methods previously evaluated outperformed more statistical methods [13]. Fickett attributes the neural networks performance either to the factors considered by the network or to the integration method employed by the network [13].

1.2.2 Similarity Searches

Sequence similarity searches or lookup methods are based on sequence conservations resulting from evolutionarily conserved properties [13] [26]. Many lookup methods use alignment scores as measures for determining protein function or protein coding potential. Two types of alignment methods are used by lookup methods [26]:

Local Alignment – Finds the region of greatest similarity between two sequences. Differences outside of the region of greatest similarity are ignored.

Global Alignment – Requires the sequence alignment to start at the beginning of each sequence and to continue to the end of each sequence.

These alignment methods are used in rigorous, similarity-search algorithms (such as the Needleman-Wunsch and Smith-Waterman algorithms) and in rapid, heuristic algorithms (such as the widely used FASTA and BLAST algorithms) [26].

Rigorous algorithms calculate the optimal similarity score between two sequences while rapid heuristic algorithms do not guarantee an optimal score for every element in a sequence library [26]. Although they do not guarantee optimal scores, rapid heuristic methods are five to fifty times faster than rigorous algorithms like the Smith-Waterman algorithm. The faster execution time of rapid algorithms are due to the smaller number of potential alignments analyzed by rapid techniques [26].

Two main rapid heuristic methods, BLAST and FASTA, can be characterized as follows [26]:

- BLASTP
    - Most widely used program for rapid sequence comparison.
    - Accurately estimates statistical significance of similarity scores.
    - Looks at regions with conserved amino acid triplets.
    - Uses a discrete finite automaton to recognize substitutions.
- FASTA
    - Calculates optimal scores and accurately estimates significance of scores.

- With the above improvements, FASTA performs better than BLASTP and nearly as well as Smith-Waterman methods.
- Looks at regions with high pairwise and single element alignment similarities.
- Uses Smith-Waterman algorithm to produce the final sequence alignments.

Similarity searches are used mostly for determining the functionality of proteins. They can be used in combination with template methods for annotating genomes. The first pass annotation of the *Drosophila melanogaster* genome was performed using similarity searches as well as gene finding software [27] [28]. Greater weight was placed on results from the similarity methods than the gene finding software [28]. Though similarity methods provide insight into the functionality and identity of new genes, rarely expressed genes may be difficult to locate with lookup methods.

1.3 Information Theory and Gene Identification

Historically, the application of information theory to genetic analysis began in the nineteen seventies [29] [30] [31]. Between 1970 and 1977, in an attempt to quantify and convey the complexity of DNA, methods were developed for estimating information, redundancy or divergence parameters for DNA sequences [29]. These efforts did not prove completely successful. After a ten year hiatus, the increase in genomic data encouraged renewed interest in the use of information theory constructs in the study of genomics. This second research period began in 1987 and continues to the present. In this present period, techniques from the field of signal processing (such as auto correlation analysis, Fourier transform, and random walks) have been used in the informational analysis of genetic sequences [29]. Discovering the existence of long-range correlations in DNA sequences proved to be a significant result of information-based analysis of genetic sequences. Mutual information, an information theory measure, has also been used to detect long-range correlations in nucleotide sequences [29]. Other information measures, such as entropy based measures, have been used in recognition of DNA patterns, classification of genetic sequences, and various other computational studies of genetic sequences [29] [32] [33] [2] [34] [35] [36] [37] [38] [39] [40] [41] [42] [10] [43].

1.3.1 Information Theory Framework for Biological Information Processing

Roman-Roldan et al. suggest that living beings can be characterized by their information processing ability and hence information based analyses can be used in their study [29]. The use of information theory in genetic data analysis requires redefinition of the genetic system as an information system. According to Roman-Roldan et al., "the processing of biological information has an artificial parallel: the processing of information by computers." This communication view of the genetic system was also suggested by May [11]. Viewing protein synthesis as an information processing system, allows nucleotide sequences to be analyzed as messages without considering the physical-chemical elements for information processing [29]. Transfer of biological information can be modeled as a communication channel with the DNA sequence as the input and the amino acid sequence which forms protein as the channel output [29]. The communication channel view suggested by Roman-Roldan et al., depicted in Figure 1.1 slightly differs from the view presented by May [11]. Instead of designating

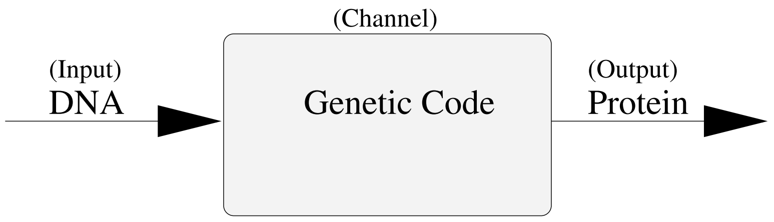


Figure 1.1: Information Theory Based View of Protein Synthesis (Roman-Roldan, et al.)

the amino acid sequence as the channel output, May's model, depicted in Figure 1.2, defines the messenger RNA (mRNA) as the output of the communication channel and incorporates a decoder that translates the mRNA into protein forming amino acid chains.

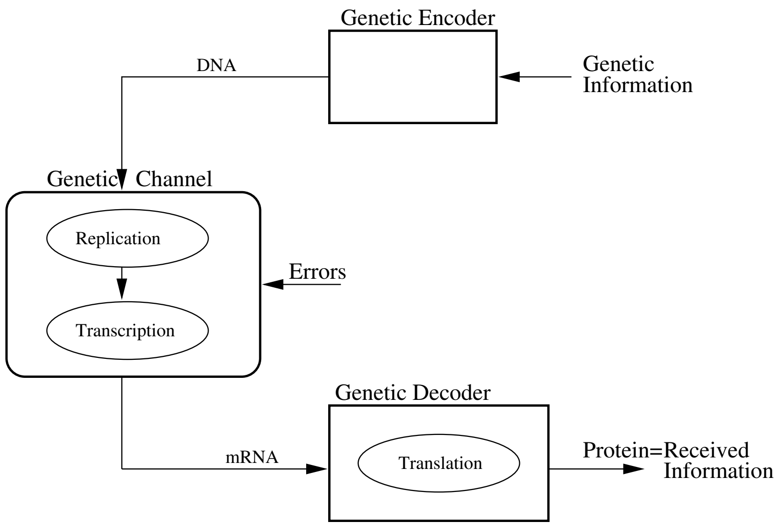


Figure 1.2: Central Dogma of Genetics as a Coding System (May, et al.)

Roman-Roldan et al. defines the genetic information source as an ergodic source that generates messages from a finite alphabet. An ergodic source is a source that, using a random selection criteria, generates typical messages and atypical messages. Typical or statistically homogenous messages are generated with probability close to one while atypical messages are generated with probability close to zero [29]. Roman- Roldan et al. define the genetic information source with the following parameters:

- Genetic Alphabet: B=[A, C, G, U], where the members of the alphabet represents adenine, cytosine, guanine, and uracil, respectively.
- p(A) + p(C) + p(G) + p(U) = 1
- Genetic message source is modeled as a Markov source (bases in a message are not independent) with a stochastic distribution matrix $[p(B_i|B_j)]$, $\sum_i p(B_i|B_j) = 1$. The Markov source is assumed to be stationary and ergodic.

Roman-Roldan et al. designate the genetic code, the process of mapping codons to amino acids, as the transmission channel through which DNA is transmitted and protein is received. May's definition of the genetic channel differs from that given by Roman-Roldan et al. May defines the genetic channel as the DNA replication and transcription process during which errors are introduced into the nucleotide sequence [11]. But both May and Roman-Roldan et al. assume the transmission channel to be stationary and memoryless. If the genetic channel is noiseless, or free of genetic mutations, in Roman-Roldan et al.'s model the input/output probabilities are specified as follows [29]:

$$p(A_i/B_1, B_2, B_3) = \begin{cases} 1, & if \quad (A_i/B_1, B_2, B_3) \\ & \text{is part of the genetic code} \\ 0, & otherwise \end{cases} \tag{1.1}$$

According to Roman-Roldan et al. the success of using information theory in genetic analysis is directly linked to the quality of the first order approximation of DNA, the string model. The string model of DNA represents DNA as a string of letters (the nucleotide bases) in the order they appear in the nucleic acid molecule [29]. Many of the recent information theory based genetic analysis techniques use this first order approximation of DNA. One such technique, developed by Schneider (who also defines molecular processes based on a communication theory framework [2]), is discussed in the following section.

1.3.2 Information Based Models

As previously mentioned, several researchers use information based measures in the analysis and classification of genetic sequences and processes. This section reviews Schneider's use of entropy or uncertainty to analyze and identify binding sites on nucleic acid sequences [10, 44]. While Schneider's work illustrates a direct application of information theory to the genetic analysis problem, Eigen's work, reviewed briefly in this section, uses information theory principles to study the origin of genetic information in living systems [12].

Information Theory in Binding Site Analysis

The information based model, used by Schneider et al. [44], is based on the information theory concept defined by Shannon [44]. Shannon defined one "bit" as the amount of information needed to distinguish between two equally probable symbols. To distinguish M symbols which have equal probability of occurrence, we need $log_2 M$ bits [2]. For a given signal, a series of symbols or bases, Schneider et al. define an associated information based measure R, a variable which indicates the average (in an information theory sense) amount of information in the signal. R is a measure of the information gained and is measured in bits per symbol [2]:

$$R = H_{before} - H_{after} \tag{1.2}$$

where

$$H = -\sum_{i=1}^{M} p_i \log_2 p_i \tag{1.3}$$

where p_i is the probability of each symbol i.

Schneider et al. use these information theory measures to analyze nucleotide sequences and identify highly conserved regions. In [10] and [44] Schneider et al. develop methods for analyzing the informational content of binding site sequence groups and individual binding site sequences, respectively. Binding sites are regions on DNA and RNA sequences to which macromolecules such as repressers, polymerases, and ribosomes, bind.

In [44] Schneider et al. analyze *E. coli* binding site groups using two information based measures derived from the Shannon entropy, H:

- $R_{sequence}$ - A measure of the information in the binding site sequence patterns.
    - Describes how different the binding site sequences are from all the other genomic sequences.

- The value of $R_{sequence}$ should be related to the binding interaction between the macromolecules that bind to the sites and the binding sites themselves.
- Measured in bits per site.

- $R_{frequency}$ - The amount of information needed to locate the binding site, given that the binding site occurs with a certain frequency in the genome.
  - Measures the amount of information needed to locate binding sites (i.e. information necessary for site distinction).
  - Value depends on the size of the genome and on the number of binding sites in the genome.
  - The $R_{frequency}$ of a less occurring binding site would be greater than the $R_{frequency}$ of a more prevalent binding site.
  - Measured in bits per site.

Using genetic sequences which contain known binding sites, Schneider et al. calculated $R_{sequence}$ as follows [44]:

1. Aligned binding site sequences by the zero base or the first base of the initiation codon.
2. From the aligned sequences, formed the frequency table $f(B, L)$, where $f(B, L)$ represents the frequency of base
$$B = [(A)denine,\quad (C)ytosine,\quad (G)uanine,\quad T(hymine)\quad or\quad U(racil)]$$
at sequence position L.
3. Using Shannon's entropy, Equation 1.3, and the calculated positional frequency table, they formed $H_s(L)$, the positional entropy:
$$H_s = -\sum_{B=A}^{T} f(B, L) \log_2 f(B, L) \quad bits/base \tag{1.4}$$

The positional entropy takes on values between zero (if only one base appears) and two (if all four bases are equiprobable) bits per base.

4. Positional $R_{sequence}$ is then defined as:

$$R_{sequence}(L) = H_g - H_s(L) \quad bits/base \tag{1.5}$$

where $H_g = H_{genome}$ is close to two bits per base for the *E. coli* organism used in Schneider et al.'s study.

5. Assuming that the frequency at each position is not influenced by the frequency at another position, $R_{sequence}$ is:

$$R_{sequence} = \sum_L R_{sequence}(L) \quad bits/site \tag{1.6}$$

Results of the $R_{sequence}$ calculations can be graphically displayed using sequence logos. Sequence logos show base conservation at various positions and regions within the sequence [45]. Each position represents the conserved base or bases in bits per symbol. Hence, a completely conserved symbol would be two bits high at the position of conservation. Highly conserved sequence regions (locations with informational spikes) indicate areas of key structural contacts and regions of genetic interactions within a nucleotide sequence [44].

The calculation of $R_{frequency}$ is also based on entropy measures similar to those used to calculate $R_{sequence}$. The details for the calculation of $R_{frequency}$ can be found in [44]. $R_{sequence}$ and $R_{frequency}$ serve as quantitative tools for studying how proteins locate their respective binding sites among non-binding site sequences. Schneider et al. found that these two genetic measures are similar for the binding sites evaluated in their work [44].

Schnedier et al. calculated $R_{sequence}$ and $R_{frequency}$ for the ribosome binding sites of 149 *E. coli* sequences. The highest informational peak in $R_{sequence}(L)$ occurred at the initiation codon. The second highest peak occurred at the Shine-Dalgarno site. For the ribosomal binding site, $R_{sequence} = 11.0 \quad bits/site$ [44]. Eleven bits translates into 5.5 bases. This suggests that a window of six bases should be sufficient in ribosome binding site recognition. The ribosome binding site contains more than six bases. Assuming the ribosome binding site is thirty bases long, if each position is equally conserved (contributes the same amount of information) then each base would contribute less than 0.5 bits of information. But, all bases are not equally conserved in the ribosome binding site. Hence the 11.0 bits of information required to distinguish a ribosome binding site are not evenly distributed. Since these 11.0 bits are not necessarily evenly distributed across the ribosomal binding site, a six base window may be insufficient for distinction unless the ribosome has some type of "memory" mechanism.

According to Schneider et al., the genetic measure $R_{sequence}$ does not reveal anything about the physical mechanism employed by the macromolecule in binding site recognition. Yet, Schneider was able to use the informational measure $R_{sequence}$ in recognizing individual binding sites based on their individual informational content $R_i(j)$ [10].

Using similar methods as previously described, Schneider created $R_{iw}(b,l)$, a four by L individual information weight matrix [10]. Using $R_{iw}(b,l)$ and the nucleotide sequence, Schneider calculated the individual information content $R_i(j)$ of the j-th test binding site sequence. This value was then compared to a histogram derived from sample binding sites. The expected value of the histogram is $R_{sequence}$ [10]. For instance, for *E. coli* ribosomal binding sites, $R_i = 8.68 \pm 3.42$ bits [10]. This technique is used to evaluate and search for new binding sites. Schneider et al.'s information based evaluation of binding sites led to two notable discoveries [10]:

- The consensus sequence (or the most "perfect" sequence) is improbable.

- The method proves that there exists an evolutionary relationship between changes or variations of specific control points and the overall cellular control mechanism.

These two ideas indicate that the genetic translation system (most likely the genetic system as a whole) permits, if not requires, some degree of error. Therefore it must provide some method of error detection and error correction.

Information Theory and Genetic Evolutionary Analysis

Eigen [12] evaluates evolution based on a living system's informational capacity. According to Eigen, the period in which non-life (chemistry) transitions to life (biology) is the period during which genetic information is generated. The living system which houses this genetic information is defined as a "complex adaptive system [capable] of functional self-organization based on the processing of information [12]." This information is generated by a feedback loop through the biological mechanism of replication and the evolutionary mechanism of selection.

Eigen asserts that if reproduction is the foundation for information conservation and if reproduction causes natural selection then there must exist an error threshold for reproduction. Above and below said error threshold, information is lost [12]. Only near the error threshold of reproduction will there exist a large population of viable variations or mutants. This mutant distribution or "quasi-species" have a defined consensus sequence. Their sequences are similar but non-identical [12]. The mutants that form this set of quasi-species are the ones which survive, hence resulting in evolutionary flexibility.

Eigen's notion of quasi-species and reproductional error thresholds provide a framework for narrowing our focus from an informational view of genetic systems to a coding theory view of the genetic system. The idea that quasi-species are a set of viable mutants derived from a consensus sequence parallels coding theory's notion of a set of valid codewords which are related yet have variations in their individual sequences. Like quasi-species there exists an error detection and error correction threshold for codes. Much like the reproductional error threshold, when errors in an encoded sequence surpass the coding error threshold information is lost.

Eigen further suggests that the genetic information, DNA, has error correcting capabilities and that the complementary interactions found in the DNA molecule provided for an encodable alphabet. The information space concept, or sequence space, developed by Eigen maps nucleic acid sequences to a discrete point space [12]. The distance between the points (sequences) in the sequence space is equal to the number of positions in which the sequences differ from one another [12]. Eigen's sequence space can be paralleled to a decoding sphere that is composed of n-symbol sequences that are located around an n-symbol codeword [46]. The sequence distance concept is equivalent to the Hamming distance concept in coding theory [46, 20].

Eigen and Schneider's work leads us towards a coding theory framework for the analysis of genetic information.

1.4 Coding Theory and the Gene Identification Problem

Battail [1] argues, similar to Eigen, that for Dawkins' model of evolution to be tractable, error-correction coding must be present in the genetic replication process. In Dawkins' model DNA and RNA are replicators housed in phenotypes or survival machines. The phenotypes are controlled through the genetic code and are subject to natural selection. In Dawkins' evolutionary model, survival of a replicator implies reliable replication [1]. According to Battail, proof-reading, a result of the error avoidance mechanism suggested by genome replication literature, does not correct errors present in the original genetic message.

Only a genetic error correction mechanism can guarantee reliable message regeneration in the presence of errors or mutations due to thermal noise, radioactivity, and cosmic rays [1].

Battail further asserts that the need for error protection becomes obvious when one considers that the number of errors in a k-symbol message that has been replicated r times is comparable to the number of errors in an unreplicated $r * k$-symbol message. For a given error rate, the number of times an organism undergoes replication approaches an infinite number. Hence for a message to remain reliable within an organism's life cycle (not to mention evolutionary information transmission which occurs over thousands of years) the message must have strong error protection [1]. Battail points out that if there exists a minimum Hamming distance d between codewords, then almost errorless communication is possible if and only if the following holds:

$$p * n < d/2 \tag{1.7}$$

where p is the error probability for the channel and n is the length of the codewords. If we take n to be the length of the gene or a portion of the gene, minimum distance decoding may be used to produce a near errorless rule [1]. The use of minimum distance decoding model was explored by May [11], where a possible block code for ribosomal binding site sequences in *E. coli* was tested. Eukaryotes' tendency to evolve towards increasing complexity may parallel the connection between increasing word length and increasing reliability, which is stated in the fundamental theorem of channel coding [1]. The fundamental theorem of channel coding states that coding rates that are below the channel capacity result in arbitrarily small probabilities of error ($\lambda^n \to 0$) for sufficiently large blocks lengths, n [47].

The survival of an organism necessitates the existence of a reliable information replication process. Therefore error-correcting codes must be used in replication or in another process of information regeneration that precedes replication [1]. Battail also suggests that genetic information undergoes nested encoding, where the result of a previous encoding process is combined with new information and encoded again. The more important genetic information is assumed to be in the primary coded message [1].

Battail's supposition regarding nested coding mirrors coding theory's concept of concatenated codes which are also called nested codes [46]. Based on Battail and Eigen's works, the communication view of the genetic system proposed by May [11] can be modified as follows:

- The replication process will represent the error-introducing channel.
- The transcription process will be part of the first level decoding process of a nested decoder.
- Translation initiation will be part of the second level decoding process of a nested decoder.
- Translation elongation and termination will be the third and final level of decoding.

Figure 2.1 depicts these modifications.

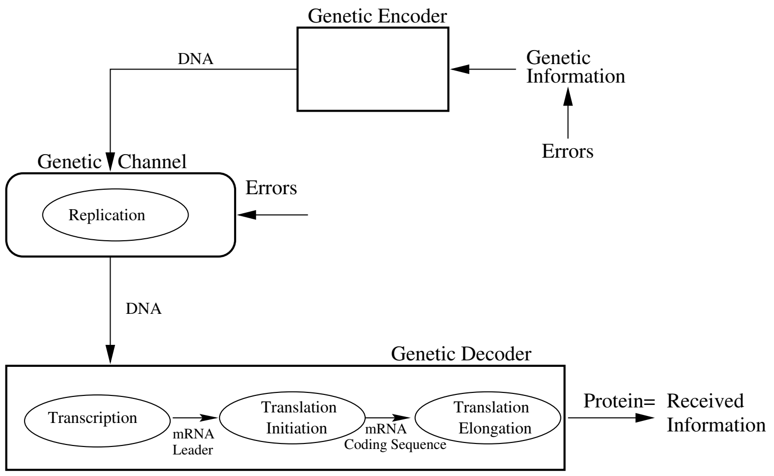


Figure 1.3: Modified Coding Theory View of the Central Dogma of Genetics

1.4.1 Coding Theory Based Models

Battail makes a plea for increased research for the purpose of identifying the error-correcting process proposed in [1]. Though there is little known research into error-correcting models for genetic processes (beside experimental work by May et al. [11] [4] [5] and Schneider's proposed coding model for molecular machines [2]), there is some research into coding theory based approaches to analyzing genetic sequences [3][48][49][50][51].

1.4.2 Coding Theory and DNA Computing [3]

Kari et al. used circular codes to define heuristics for constructing codewords for DNA computing applications. In DNA computing, the information storage capability of DNA is combined with laboratory techniques that manipulate the DNA to perform computations [3]. A key step in DNA computing is encoding the problem in the DNA strand. The challenge is to find codewords for encoding that do not form undesirable bonds with itself or other codewords used or produced during the computational process. Kari et al. used coding theory to define rules for constructing "good" codewords for DNA computing.

Use of Coding Theory in Reading Frame Identification

Arques et al. statistically analyzed the results of 12,288 autocorrelation functions of protein coding sequences. Based on the results of the autocorrelation analysis, they identified three sets of circular codes X_0, X_1, X_2 which can be used to distinguish the three possible reading frames in a protein coding sequence [48].

A set of codons X is a circular code, or a code without commas, if the code is able to be read in only one frame without a designated initiation signal [48]. Crick et al. originally introduced the concept of codes without commas in the alphabet A, C, G, T. It was later successfully addressed and extracted over the alphabet R, Y, N [48]. Arques et al. successfully defines a circular code over the A, C, G, T alphabet. They were able to use the three sets of circular codes to retrieve the correct reading frame for a given protein sequence in a thirteen base window. The three circular codes are described using a flower automaton. Arques et al. have used their coding based model to analyze the Kozak's scanning mechanism for eukaryotic translation initiation and other models of translation [48].

Coding Theory Based Sequence Analysis

Stambuk also explored circular coding properties of nucleic acid sequences [49] [50]. His approach was based on the combinatorial necklace model which asks: "How many different necklaces of length m can be made from bead of q given colors [52, 49]." Using $q = [A, C, G, T]$ and $q = [R = Purine, Y = Pyrimidine, N = R \quad or \quad Y]$ Stambuk applied the necklace model to genetic sequence analysis, enabling the use of coding theory arithmetic in the analysis of the genetic code [49].

Though Stambuk did not use error control coding in his analysis, his work provided important insight into the structure of DNA sequences [49, 50]:

- Non-protein coding DNA contains properties corresponding to natural language; protein coding DNA has properties that are characteristic of coded language structures.
- A binary nucleotide mapping and a corresponding Gray code mapping can be defined based on chemical properties of the bases. Binary mapping incorporates complementarity of DNA and Gray code mapping ensures error minimization during the translation and transcription process.
- The Hamming distance measure can be used to express the difference between different codon and different amino acid positions.

The binary nucleotide coding (corresponding to non-coded genomic patterns) and the Gray code mapping (corresponding to coded genome patterns) defined in Stambuk's work may prove useful in "the extraction/decoding of the programming language of DNA and RNA strings [49]." May et al.'s coding based modeling approach will be presented in chapter two.

1.5 Document Organization

The chapter which follows gives an overview of previous research by May [11][4][5][6] which is the foundation for this present work. Chapter three discusses the convolutional coding technique used in this research: binary and base five table-based convolutional codes. In chapter four the use of genetic algorithms to find base five table-based convolutional codes for translation initiation is presented. Chapter five presents binding based binary codes for translation initiation and chapter six applies the coding theory concepts to other prokaryotic organisms. This document concludes with chapter seven which summarizes the major findings and contributions of this work, including a discussion of future research directions based on this work.

Chapter 2

Theoretical Background

The rapid advances in both genomic data acquisition and computational technology has encouraged development and use of engineering based methods in the field of genetic data analysis. Techniques from engineering fields such as Signal Processing[24, 22, 53], Machine Learning[25] and Information Theory[29, 35, 42, 10], and various statistical methods[21, 16, 17, 18, 14, 19] are now being heavily researched for use in gene identification. Several researchers are encouraging the use of coding theory, specifically error-correction coding, in analyzing genetic data [1]. A goal of current work in this context is to use coding theory based analysis to determine whether regions of the specified genome are protein-producing sequences.

From previous work in protein annotation and gene identification, we make several key observations. Redundancy occurs naturally within RNA and DNA sequences [9]. Mutations or errors are present within the genome of an organism. Ribosomal binding sites (translation initiation sites) appear to evolve to functional requirements rather than to genetic sequences that produce the strongest binding site [10]. Viable mutants, or imperfect sequences, have error rates near an error threshold assuring the organism's evolutionary flexibility [12]. Therefore, survival and evolution of an organism necessitates errors, and hence the existence of a genetic error correction mechanism [1]. Our final observation is that the ribosome maps or decodes a fixed length nucleic acid signal (codon) to specific information (amino acid). From these observations, it is shown that the transmission of genetic information can be viewed as a biological, cellular communication system that employs some method of coding to recognize valid information regions and to correct for “transmission” errors. Given that messenger RNA is viewed as a noisy encoded signal, the principal hypothesis of this work is that it is feasible to use principles of error control coding theory to interpret the genetic translation initiation mechanism.

This chapter begins with an overview of the theoretical background of this work. Previous research and results are presented in Section 2.2.

2.1 Theoretical Background

In a communication system, error-control coding techniques are used to compensate for errors that occur during transmission of information. Error control is accomplished by introducing redundancy into the original information sequence through a well-defined encoding algorithm[54, 55]. Therefore, there are more symbols in the transmitted sequence than in the original sequence.

The use of coding theory and its techniques stems from the need for error control mechanisms in a communication system. In an engineering communication system, digitized information is encoded by the channel encoder and prepared for transmission (modulation). The information is then transmitted through a potentially noisy channel where the transmitted information may be corrupted in a random fashion. At the receiving end, the received message is prepared for decoding (demodulation) and then decoded by the channel decoder[54, 20]. The decoding process involves removal and perhaps correction of errors introduced during transmission. The decoding mechanism can only cope with errors that do not exceed its error correction capability.

The relationship between the coding process and the protein translation process may not be an obvious one. We note that, similar to error-control encoded information, redundancy naturally occurs within RNA and DNA sequences. The existence of redundant constructs like tandem repeats, Shine-Dalgarno sequences, the Pribnow box, and the TATA box leads us to believe that biological, cellular communication systems use redundant signals to protect the genetic message from errors. Hence, the genetic system must employ some method of coding to recognize valid information regions within a nucleotide sequence and correct for "transmission" errors such as mutations. Based on the principles of the engineering communication system, we develop a communication view of genetics using the central dogma of genetics: information stored in redundant DNA sequences is replicated, transcribed into redundant messenger RNA, and ultimately translated into proteins[56, 9]. The DNA sequence is the output of a genetic encoder and the input into an error-introducing channel or the replication process. The biological process that corresponds to the encoder is not yet known. A nested encoding (concatenated coding) model is assumed, where the most vital information receives the greatest protection [1]. The replicated DNA is decoded by a three-level genetic decoder. Transcription is the first level of decoding, followed by translation initiation, and finally translation elongation plus termination. Figure 2.1 illustrates this communication view of the genetic process.

Since coding produces encoded blocks based on present and past information, it seems reasonable to assume that genetic operations such as initiation and translation may involve "decisions" which are based on the immediate neighborhood of a codon. This would allow error correction and other related functions. This work uses coding theory tools and techniques to develop a computational model for identifying and understanding protein translation initiation sites.

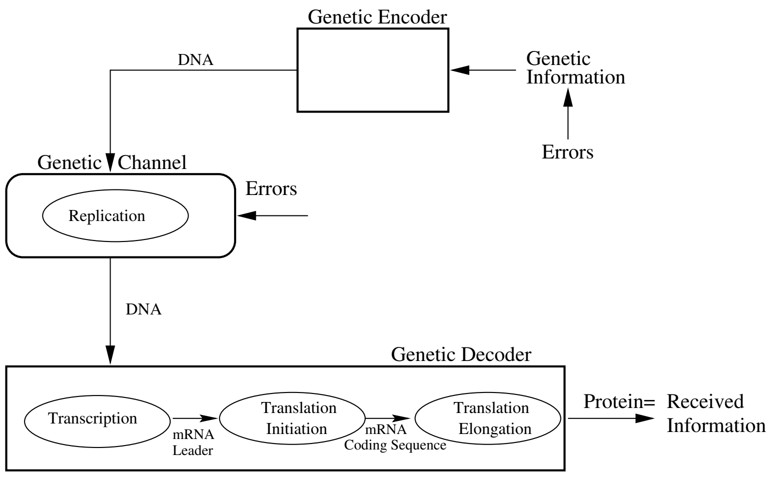


Figure 2.1: Communication View of the Central Dogma of Genetics

2.2 Previous Code Models for Translation Initiation[4][5][6]

As mentioned earlier, the encoding mechanism used in the genetic encoder is unknown. Therefore, one does not know the exact mechanism employed by the genetic decoder. By analyzing key elements involved in initiating protein translation, it is hoped that we will gain insight into a possible decoding scheme used in the initiation of translation in prokaryotic organisms. The key elements considered are: the 3' end of the 16S ribosomal RNA, the common features of bacterial ribosomal binding sites (such as the existence and location of the Shine-Dalgarno sequence), and RNA/DNA base-pairing principles. Previous work explored a block encoding/decoding model and convolutional encoding/decoding model for the translation initiation system [11][4]. Assuming an encoding method, the corresponding decoding algorithm was designed using the 16S ribosomal RNA.

The coding alphabet must be derived from a finite field as in binary codes. Using base pairing, wobble pairing, and translation initiation information [9] the RNA bases were mapped to the field of five as follows: Inosine(I) = 0, Adenine(A) = 1, Guanine(G) = 2, Cytosine(C) = 3, and Uracil(U) = 4. Multiplication and addition are modulo five. The

RNA bases were defined so that in modulo five addition the sum of bases that form hydrogen pairs is zero. These definitions were used to construct the block code and convolutional code models for the the protein translation initiation process.

2.2.1 Error Control Coding Based Methods

Previous work experimentally analyzed the viability of the existence of an error-correction mechanism in translation initiation. Two error-control coding based models were explored: a block code model and a convolutional code model [11]. References to conference proceedings describing these models are [4] [5].

Block Code Model

In the block code model, the genetic encoder is modeled as an (n, k) block code whose output is a systematic zero parity check code [54] [11]. Codewords of length $n = 5$ and $n = 8$ were developed based on the last thirteen bases of the 3' end of 16S ribosomal RNA (which contains the hexamer complementary to the Shine-Dalgarno sequence [9]) and the proposed encoder model. The model employed a minimum distance decoder to verify the block coding model for translation initiation.

The *E. coli* K-12 strain MG1655 sequence data (downloaded from the NIH ftp site: ncbi.nlm.nih.gov) was used to test the model. Figure 2.2 shows the resulting mean minimum distance by position for the (5,2) block code model. The smaller the value on the vertical axis, the stronger the bond formed between the ribosome and the mRNA. Zero on the horizontal axis corresponds to the alignment of the first base of a codeword with the first base of the initiation codon.

As Figure 2.2 illustrates there is a significant difference among the translated, hypothetical and the non-translated sequence groups. For the translated and hypothetically

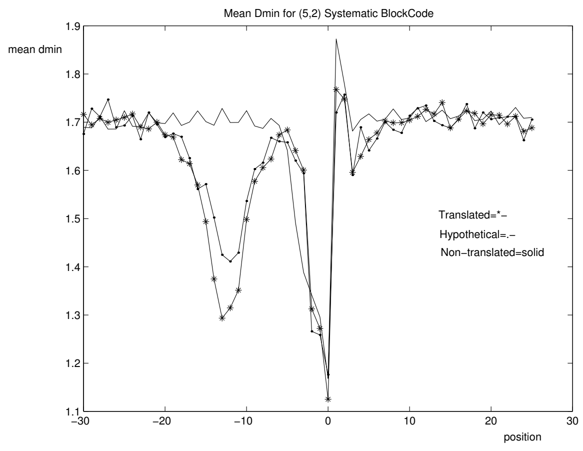


Figure 2.2: Results of Minimum Distance Block Decoding Model for (5,2) Code

translated sequence groups, a minimum distance trough occurs between the -15 and -10 regions. All the sequence groups in the (5,2) model achieve a global minimum mean distance value in the -5 to 0 region. The -15 to 0 region contains large synchronization signals which can be used to determine valid protein coding sequences or frames. There are also smaller synchronization signals outside the -15 to 0 region which seem to oscillate with a frequency of three. The results of the longer (8,2) block code model (presented in [11]) illustrate the effect of two or more codons while the (5,2) block code model is affected by at most two codons. The block code method distinguishes between translated sequence groups and non-translated sequence groups. The oscillations present in the mean minimum distance plot suggest that the leader sequence of the mRNA contains synchronization information used by the ribosome to lock on to the correct reading frame. The results of the block code model suggest that it may be possible to design a coding theory based model that can distinguish between protein coding and non-protein coding genomic sequences by "decoding" the leader region of the mRNA.

Convolutional Code Model Based Analysis of Genetic Sequences

The second error-correcting coding model previously investigated was based on the principle hypothesis that the messenger RNA (mRNA) sequence can be viewed as a noisy, convolutionally encoded signal. The ribosome was functionally paralleled to a table-based convolutional decoder. The 16S ribosomal RNA (rRNA) sequence was used to form decoding masks for table-based decoding. Convolutional coding produces encoded blocks based on present and past information bits or blocks. The model is based on the assumption that genetic operations such as initiation and translation may involve “decisions” which are based on immediate past and immediate future information. This would allow error correction and other related functions. The convolutional code model viewed the ribosome as a mechanism with memory, which differs from Schneider’s idea of macromolecular machines without memory [2]. Evaluating the messenger RNA as convolutionally encoded data, allowed the model to capture the inter-relatedness between the bases in a mRNA sequence.

Figure 2.3 shows the frequency of the most frequent distance pattern among all possible two-symbol distance patterns $d_i d_j$, where distance values range from zero to four. The horizontal axis indicates position, with zero corresponding to the alignment of the coding mask with the first base of the initiation codon. The vertical axis indicates frequency (0.04 corresponds to four percent). The expected frequency of occurrence for a random, two-symbol distance pattern is four percent.

As shown in Figure 2.3, the convolutional code model was able to distinguish between translated and non-translated sequence groups. The distinction among hypothetical and translated groups is also evident. In the convolutional code model, the higher frequency of occurrence values present in the hypothetical group (when compared to the translated

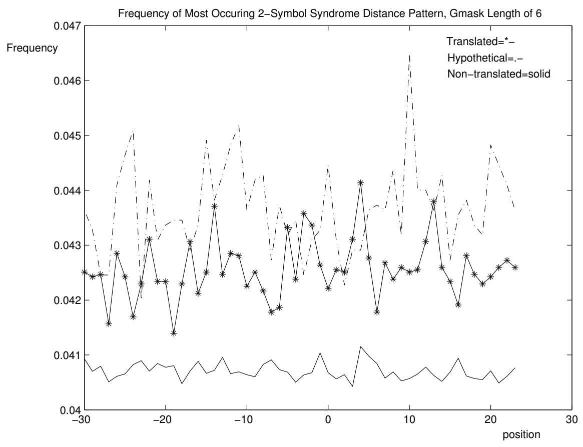


Figure 2.3: Frequency of Two-Pattern Syndrome Distance Values

group) may be a result of biasing introduced through hypothetical sequence identification methods, which are based on finding statistically significant patterns within possible reading frames. The convolutional code model indicated greater information or occurrence of significant activity in the area spanning the -15 to 0 region. The Shine-Dalgarno sequence is located within this region [9].

The preliminary results of this model suggest that it may be possible to design a convolutional coding based heuristic for distinguishing between protein coding and non-protein coding genomic sequences by "decoding" the mRNA 5' untranslated leader region.

Analysis of Coding-Based Models

Three issues were critical to analyzing the effectiveness of each model.

1. Recognition of regions within the mRNA leader sequence
2. Distinction between translated and non-translated sequence groups
3. Indication and recognition of the open reading frame construct

As Figure 2.2 illustrates the block code model indicated a significant difference between the translated, hypothetical and the non-translated sequence groups. For the translated and hypothetically translated sequence groups, a minimum distance trough occurred between the -15 and -10 regions. All the sequence groups in the (5,2) model achieved a global minimum mean distance value in the -5 to 0 region. The -15 to 0 region contained large synchronization signals which could be used to determine valid protein coding sequences or frames. There were also smaller synchronization signals outside the -15 to 0 region which seem to oscillate with a frequency of three.

Figure 2.3 indicates that for the convolutional code model, the hypothetical group contained the greatest frequency of occurrence values, followed by the translated and the non-translated group. Prior to the zeroth position (position of the initiation codon), the highest frequency value for a given distance pattern occurred around -14 for translated regions. There was a distinction in the frequency of occurrence of two-symbol patterns between the translated/hypothetical group and the non-translated group. The two-pattern frequency analysis for the syndrome values was used as a preliminary indicator to test whether syndrome values can correlate to information. The results suggested that the translated group contains more two-pattern syndrome values than the non-translated. The distance in pattern frequency percentages between translated and non-translated may vary for greater pattern lengths.

Both models distinguished translated sequence groups and non-translated sequence groups. They both also indicated the existence of key regions within the mRNA leader sequence. But, the block code model seemed to recognize the ribosomal binding site (the location of the Shine-Dalgarno sequence) more readily than the convolutional code model. The block code model also indicated the existence of a reading frame synchronization construct more so than the convolutional code model. Additional results for longer block codes and results for the longer gmask(twelve-base masks) are presented in [11].

2.2.2 Block Code Based Maximum-Likelihood Classifier

Based on the results of the block code model, a maximum likelihood classification system was designed. The system classified individual mRNA sequences as translated or non-translated based on minimum distance values in the -15 to -11 window. The -15 to -11 window provided the greatest distinction between the mean minimum distance values of translated and non-translated sequences in *E. coli* K-12.

The maximum-likelihood classifier was based on

- A measurable classification parameter s,
- The statistical nature of selected training set $P(w_i|s)$, and
- Prior knowledge of the occurrence of each classification class $P(w_i)$.

From the above, the maximum-likelihood discrimination function for the classifier emerged:

$$P(w_i|s) = \frac{P(s|w_i) * P(w_i)}{P(s)} \tag{2.1}$$

where,

$$i = (Translated, Non - translated) \tag{2.2}$$

and

$$P(s) = \sum_{j=1}^{N_{class}} P(s|w_j) * P(w_j) \tag{2.3}$$

The variable N_{class} is the number of classification classes. Since $P(s)$ is the same for all classification classes, we can (for the purpose of classification) simplify the maximum-likelihood discrimination function, Equation 2.1, to

$$P(w_i|s) = P(s|w_i) * P(w_i) \tag{2.4}$$

In the discrimination function, the value of the classification parameter s is the sum of the positional Hamming distance values within the -15 to -11 window:

$$s = \sum_{p=-15}^{-11} D_{avg_p} \tag{2.5}$$

The positional Hamming distance value D_{avg_p} is the average of the N lowest Hamming distance values at position p, where $N = 0.10$ or ten percent of the total number of codewords in the codebook. Using the number of $A-T-G$ codons and the number of genes in the *E. coli* genome, two different prior conditions were calculated and used as prior probabilities, $P(w_i)$:

1. P(Translated)=9.39%

   P(Non-Translated)=90.61%

2. P(Translated)=50%

   P(Non-Translated)=50%

Also, two different statistical distributions were used to model the probability that s occurs given we are in class w_i:

1. Cumulative Distribution Function (CDF): $P(S \leq s|w_i)$

2. Probability Distribution Function (PDF): $P(S = s|w_i)$

Testing all possible combinations of the above parameters produced four different discrimination functions, hence four different maximum-likelihood classifiers. Given the discrimination function, the maximum-likelihood classification rule was:

- Select $w_i = w_{Translated}$ if

  $P(w_{Translated}|s) > P(w_{Non-translated}|s)$

Table 2.1: Results of M-L Classifier for (5,2) Block Code

	True Positive	False Positive	False Negative	True Negative
CDF	2.51	2.28	6.88	88.33
CDF(Prior=0.5)	50	50	0	0
PDF	1.87	1.61	7.52	89
PDF(Prior=0.5)	33.95	10.14	16.05	39.86

Table 2.2: Classification Rates for (5,2) M-L Classification Systems

	Correct Classification	Incorrect Classification
CDF	90.84	9.16
CDF(Prior=0.5)	50	50
PDF	90.87	9.13
PDF(Prior=0.5)	73.81	26.19

- Else select $w_{Non-translated}$ if

  $P(w_{Non-translated}|s) > P(w_{Translated}|s)$
- Else indicate a tie occurred.

Maximum-Likelihood Classifier Results

The previously described classification system was implemented using the codebook generated from the (5,2) block code model. Table 2.1 shows the results for the four different maximum-likelihood classifiers. When using equal prior class probabilities, the PDF based classifier had a higher sensitivity than the CDF based classifier. Of the four classification systems, the PDF with equal prior probabilities seemed to perform the best, classifying translated and non-translated equally well, while maintaining a relatively low rate of incorrect classifications. Table 2.2 shows the correct versus incorrect classification rates for all four classification systems.

To improve the classification systems that use unequal prior probabilities, the block code model must produce codewords that have a greater separation than the present code. Figure 2.4 shows the non-translated and translated PDFs for the (5,2) block code.

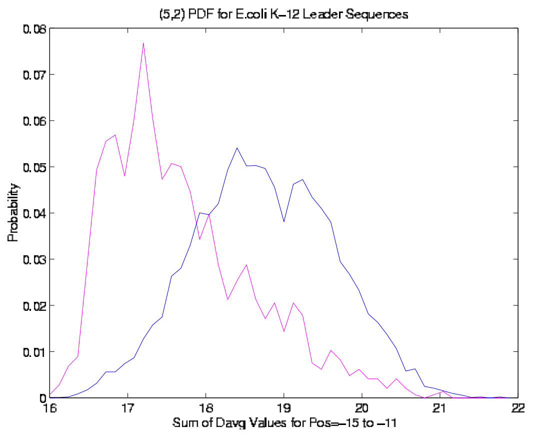


Figure 2.4: Probability Distributions for (5,2) Maximum-Likelihood Classifiers

Reducing the region of overlap could increase the sensitivity of the classifier and reduce incorrect classification rates.

The results show that the block code based maximum-likelihood classifier identified "errors" or non-translating sequences with a high degree of accuracy. From an error-control coding theory perspective, the classification system successfully detected uncorrectable errors with a high degree of accuracy. The classification system had a high specificity and used a relatively small decision window (a nine-base window). Due to the relatively small decision window, the present M-L classification system may be a promising coding-based method for identifying sequences with low translation initiation probabilities. The results of the classification systems suggest that it is highly possible to implement an error-control coding based classification for determining and possibly designing prokaryotic translation initiation sites.

Chapter 3

Table-Based Coding Systems

Previous research [11] suggested that the translation initiation system could be modeled using principles of error-control codes. Based on positive results from the sliding block code model [6, 57] and the convolutional code model [5], the behavior of the ribosome can be paralleled to that of a decoder of convolutionally encoded data.

While block codes produce encoded blocks from the present information block, convolutional coding produces encoded blocks based on present and past information bits or blocks. Evaluating the messenger RNA sequence as a sequence resulting from convolutionally encoded data, allows the model to capture the inter-relatedness between the bases in a mRNA sequence. It seems reasonable to assume that genetic operations such as initiation and translation may involve "decisions" which are based on immediate past and immediate future information; that is, not just the current set of codons under consideration but also the sequence history. This would allow error correction and other important functions. Table-based codes, developed and presented by Bitzer et al., [58], will be used to model the translation initiation system as an error-control code. Specifically, the ribosome will be modeled as a table-based decoder where error-less messenger RNA parity subsequences result in zero syndrome values when mathematically combined with the ribosomal gmask.

The following sections provide an overview of convolutional codes and a discussion on the use of table-based encoding and decoding for implementing convolutional codes for base two. Implementation of table-based codes for base five convolutional codes are developed and presented in the final section of this chapter. The chapters which follow apply table-based coding principles in the search for "good" convolutional code models for translation initiation in *E. coli* K-12.

3.1 Basics of Convolutional Coding

Convolutional coding, like block coding, is carried out over a finite field, using a set of discrete source symbols. For now, we consider the binary field, consisting of [0, 1] and the operations modulo two addition and modulo two multiplication.

As previously stated, in convolutional encoding, a n-bit encoded block at time i depends on the k-bit information block at time i and on m previous information blocks [20]. Hence, a convolutional encoder requires memory. Convolutional codes are referred to as (n, k, m) codes or (n, k) codes.

3.1.1 Encoding Methodology

A convolutional encoder is a mechanism with a k-bit input vector u_i, n-bit output vector v_i, and m memory elements. Figure 3.1 [20] illustrates a $(2, 1, 2)$ convolutional encoder, where the blocks indicate memory. Figure 3.1 shows a $k = 1$, $n = 2$, or 1/2 rate encoding scheme where a block is equal to one bit. That is, for every input bit encoding produces two parity bits. The general encoding procedure is as follows [54, 20]:

- A k-bit input block at time i, u_i, is modulo two added to the previous m input bits to form the n-bit output vector v_i.
- The most recent k input bit is shifted into the memory register and the rest of the bits in the register are shifted to the right.

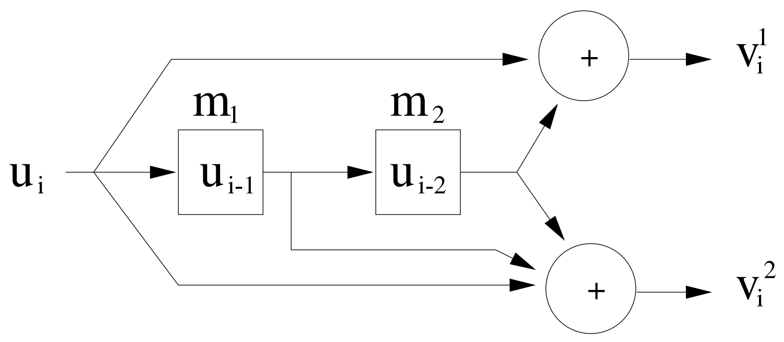


Figure 3.1: A (2,1,2) Convolutional Encoder

- The new input block is then modulo two added to the contents of the memory register to produce a new output vector.
- The process is repeated until all input data has been encoded.

A set of n generator vectors completely specify the encoder. The generators are $m+1$ bits long and indicate which elements are modulo two added to produce each bit in the output vector. For the encoder illustrated in Figure 3.1, the generator vectors are as follows:

$$g_1 = [1 \quad 0 \quad 1] \tag{3.1}$$

$$g_2 = [1 \quad 1 \quad 1] \tag{3.2}$$

The generator vectors can also be represented as generator polynomials:

$$g_1(x) = 1 + x^2 \tag{3.3}$$

$$g_2(x) = 1 + x + x^2 \tag{3.4}$$

For x^D, D represents the number of delay units. Each generator vector or polynomial is associated with one of the n output bits in the output vector v. The encoding process depends not only on the present input but also on the previous m inputs. This forms an interdependence among the input data bits.

3.1.2 Decoding Methodology

A decoder provides a strategy for selecting an estimated codeword (a codeword is the output of the convolutional encoder for a given input data block) for each possible received sequence [55]. There are various methods for decoding convolutionally encoded data. One method, maximum likelihood decoding, compares the received sequence with every possible code sequence that the encoding system could have produced. Given a received sequence and the state diagram of the encoding system that could have produced the sequence, maximum likelihood decoding produces the most likely estimate of the transmitted vector, v. The Viterbi decoding algorithm [54, 20] is a maximum likelihood decoding algorithm which uses a code trellis to estimate the transmitted vector given a received vector.

Another decoding method, syndrome decoding, uses a decoding window which consists of $m+1$ frames [54]. In syndrome decoding, the received sequence is treated like a block code and a syndrome value is generated for each received block. The value of the syndrome indicates the presence or absence of an error in the received sequence. The decoding technique we will analyze, table-based decoding, makes use of syndrome decoding techniques.

3.2 Table-Based Codes

The following sections will discuss a specific method for implementing convolutional coding: table-based encoding and decoding. All methods described for table-based encoding and decoding are based on the concepts developed and presented by Bitzer et al., in [58].

3.2.1 Table-Based Encoding

The existence of a one to one mapping between data bits and parity bits (parity bits are encoded bits) is the foundation for table-based encoding. A set of w-bit data block must correspond uniquely to a set of w-bit parity block. Parity bits are the bits generated by the encoder and they make up the output vector v. For an (n, k, m) code

$$w = n\frac{L-k}{n-k} \tag{3.5}$$

where

$$L = m + 1 \tag{3.6}$$

Table-based encoding is implemented as follows: based on the knowledge of the encoder and the parameters n, k, L we can construct an encoding table that associates each w-bit data sequence with a unique parity sequence. For binary data there are 2^w possible data sequences. Depending on the value of w, the encoding table can become extremely large. We can construct a reduced encoding table with only w data elements and the corresponding w parity elements. For the reduced encoding table each data sequence is w-bits long and contains a single bit equal to one in the *ith* position, where i goes from position one to position w. These w data sequences are the basis vectors (the fundamental vectors that can be combined to form all other vectors or sequences) for the set of all possible w-bit data sequences. For instance, the data sequences, or basis vectors, for a reduced encoding table with $w = 3$ are

$$[100 \quad 010 \quad 001]$$

The encoding masks, which are equivalent to the generator vector, are used to form the corresponding parity bits for each w-bit data sequence. In the following example a $w = 4$ bit parity sequence is generated for the encoder illustrated in Figure 3.1. For a given data sequence, parity bits are generated by ANDing the data bits with the encoding mask and XORing the results. For the data sequence *databits* =1000 and encoding mask one and two defined as

$$C_1 = [101]$$

$$C_2 = [111]$$

the parity bits $P_{1,1}, P_{2,1}, P_{1,2}, P_{2,2}$ are calculated as follows:

1 0 0 0

1 0 1

———

1+0+0 $= P_{1,1} = 1$

1 0 0 0

1 1 1

———

1+0+0 $= P_{2,1} = 1$

Shift C_1 by $k = 1$ to get next parity bit

0 0 0

1 0 1

———

0+0+0 $= P_{1,2} = 0$

Shift C_2 by $k = 1$ to get next parity bit

0 0 0

1 1 1

———

0+0+0 $= P_{2,2} = 0$

From the example above, 1100 is the corresponding parity bits for data sequence 1000.

Table 3.1: Reduced Encoding Table

Data Bits	Parity Bits
1000	1100
0100	0111
0010	1101
0001	0011

Following the same procedure we obtain Table 3.1 as the resulting reduced encoding table for the encoder in Figure 3.1.

Table-based encoding works as follows for an encoding window w data bits wide:

1. Using the reduced encoding table, process the present w data bits into w parity bits using the encoding table.

2. Shift into the encoding window k new data bits and process the data bits in the window to produce a new block of w parity bits. The new parity bits overlap the old parity block with the first $w - n$ bits of the new block. These overlapping bits are identical.

3. Repeat the encoding process until all data bits have been processed.

For table-based encoding to work, the proper encoding mask (derived from the generator vector) must be selected. The encoding mask must be chosen such that there exists a one to one correspondence between the data and parity bits. For error correcting systems, the encoding mask must produce codes that have sufficient error correcting capabilities for a given correction algorithm. This work will focus only on error-detecting codes.

3.2.2 Table-Based Decoding

Decoding tables are used to perform table-based decoding on received sequences or parity bits. A decoding table can be constructed if there exists a unique one to one mapping between data blocks and parity blocks. Therefore, table-based codes are invertible codes. The size of a decoding table for binary data would be 2^w. As in table-based encoding, we can construct a reduced decoding table which contains w elements instead of 2^w elements. Each of the parity sequences in the reduced table are w bits wide and the ith parity sequence has a single bit equal to one in the ith position, where i goes from one to w. For a reduced decoding table with $w = 2$, the parity sequences are:

$$[10 \quad 01]$$

Given a reduced encoding table, we can construct the corresponding reduced decoding table as follows:

1. Sum the x w-bit parity blocks in the reduced encoding table needed to form the parity block for the reduced decoding table.
2. Sum the x w-bit data blocks associated with the x parity blocks from the encoding table to produce the w-bit data block that corresponds to the needed parity block in step one.
3. Continue this process for all w parity block entries in the reduced decoding table.

The following is an illustration of the above method using the reduced encoding table in Table 3.1. To construct the corresponding reduced decoding table, we must find the corresponding data blocks for the following four-bit parity blocks:

$$[1000 \quad 0100 \quad 0010 \quad 0001]$$

For $parityblock = 1000$:

Table 3.2: Reduced Decoding Table

Parity Bits	Data Bits
1000	1101
0100	0101
0010	1011
0001	1010

1. $$1000 = 1100 + 0111 + 0011$$

2. The corresponding four-bit data block is:

$$1000 + 0100 + 0001 = 1101$$

3. After repeating steps one and two for the other three parity blocks, we obtain the resulting reduced decoding table shown in Table 3.2:

Given a decoding window w parity bits wide, we can decode a parity stream as follows:

- Using the encoding table, a block of w parity bits is mapped to w data bits, producing the associated w-bit data block.
- n new parity bits are shifted into the decoding window.
- From the w parity bits now in the decoding window, produce the next block of data. The $w - k$ bits at the beginning of the new data block will overlap the $w - k$ bits at the end of the previous data block.
- The above process repeats until all parity bits are decoded.

If there are no errors in the parity stream, the overlapping $w - k$ data bits will match producing zero values when exclusive-ORed bit by bit. But, if there is an error in the parity stream, the exclusive-ORing of the overlapping bits will result in non-zero values.

The results from performing the exclusive-OR operation on the overlapping data bits are called syndrome values or syndromes. A syndrome vector consists of a series of syndrome values. The syndrome vector is zero if there are no detectable errors in the parity stream (i.e exact match between overlapping bits); otherwise, for binary data, the syndrome value is one. The number of syndrome values in a syndrome vector is equivalent to the number of overlaps used to determine the vector.

3.2.3 Formation of the Gmask

The syndrome vector, which is used to detect errors in the parity stream, can be generated by repeated application of the decoding table to the parity stream. The gmask provides an efficient method for syndrome vector generation. The values that comprise the gmask are based on the codewords of the encoding system. The gmask is $w + n$ bits long. The following procedure describes how to generate the g-mask, given a decoding table.

- Consider parity streams with single bit errors in each position of the n bit parity block $[P_1 \ \ P_2 \ \ \cdots \ \ P_n]$
- Find the syndrome vector S_i for parity stream with error in parity bit P_i for $i = 1, \ \ 2, \ \ \cdots, \ \ n.$
- The gmask is formed by interleaving the n syndrome vectors generated. For instance, if $n = 3$ and $S_1 = 001$, $S_2 = 101$, and $S_3 = 110$ then the gmask can be defined as

$$g - mask = [1 \ \ 1 \ \ 0 \ \ 0 \ \ 0 \ \ 1 \ \ 0 \ \ 1 \ \ 1]$$

There will be $n - k$ gmasks for an n, k code. Once the gmask has been constructed, it can be used to calculate the syndrome vector for the parity stream. To calculate the syndrome vector using the g-mask:

- The gmask is ANDed with the first $w + n$ parity bits.

- The result is exclusive-ORed to produce a syndrome value.
- The received parity stream is shifted by n bits.
- The process is repeated until all syndrome values of the syndrome vector are produced. Each shift results in one syndrome value.

Based on the value of the syndrome vector, the received parity sequences can be used to estimate the transmitted sequence to data or used to detect errors in the transmission. The concept of a decoding mask, the g-mask, is employed in the convolutional coding model for the translation-initiation system.

3.3 Table-Based Codes for Field of Five

In early work the translation initiation system was modeled as a convolutional code over a field of five [5]. As previously stated, the coding alphabet must be derived from a finite field as in the binary code. Based on the biological characteristics of the RNA (discussed in preceding chapters) the RNA bases were mapped to the field of five as follows [59]:

- Inosine = 0
- Adenine = 1
- Guanine = 2
- Cytosine = 3
- Uracil = 4
- Multiplication represents modulo five multiplication
- Addition represents modulo five addition

The multiplication and addition operations are shown in Table 3.3 and Table 3.4 respectively [59]. The corresponding inverse operations, division and subtraction, for base five multiplication and addition are shown in Table 3.5 and Table 3.6 respectively. In Table 3.5,

Table 3.3: Field 5 multiplication operation in relation to RNA bases.

	I(0)	A(1)	G(2)	C(3)	U(4)
I(0)	0	0	0	0	0
A(1)	0	1	2	3	4
G(2)	0	2	4	1	3
C(3)	0	3	1	4	2
U(4)	0	4	3	2	1

Table 3.4: Field 5 addition operation in relation to RNA bases.

	I(0)	A(1)	G(2)	C(3)	U(4)
I(0)	0	1	2	3	4
A(1)	1	2	3	4	0
G(2)	2	3	4	0	1
C(3)	3	4	0	1	2
U(4)	4	0	1	2	3

Table 3.5: Field 5 division operation in relation to RNA bases.

	I(0)	A(1)	G(2)	C(3)	U(4)
I(0)	NaN	0	0	0	0
A(1)	NaN	1	3	2	4
G(2)	NaN	2	1	4	3
C(3)	NaN	3	4	1	2
U(4)	NaN	4	2	3	1

Table 3.6: Field 5 subtraction operation in relation to RNA bases.

	I(0)	A(1)	G(2)	C(3)	U(4)
I(0)	0	4	3	2	1
A(1)	1	0	4	3	2
G(2)	2	1	0	4	3
C(3)	3	2	1	0	4
U(4)	4	3	2	1	0

NaN is a flag indicating division by zero.

These definitions are used to construct the convolutional code model for the the protein translation initiation process in previous[11] and current work. To effectively use table-based coding techniques for the base five mapping of mRNA, encoding tables, decoding tables, gmask formation and syndrome generation concepts must be accurately developed in field five. Much of current and past work uses a base five representation for the genetic code. Therefore, the principles of binary table-based coding have been extended to equivalent principles for the field of five.

Base five table-based coding is similar to base two. Both require w basis vectors of length w for encoding and decoding. But coding in base five permits multiplication by scalar values of (0, 1, 2, 3, 4) while base two only permits multiplication by (0, 1). Also, in base five when calculating the syndrome, there will be 5^{n-k} difference vectors while in base two there will be 2^{n-k} difference vectors.

The following base five convolutional code will be used to illustrate the steps for base five table-based coding. For an $(n = 3, k = 1, m = 4)$ code

$$w = n\frac{L-k}{n-k} = 6 \tag{3.7}$$

where

$$L = m + 1 = 5 \tag{3.8}$$

The n encoding masks or generators that define the convolutional code are:

$$C_1 = [10121] \quad C_2 = [13121] \quad C_3 = [21211] \tag{3.9}$$

3.3.1 Base Five Table-Based Encoding

As in binary codes, we form the reduced encoding table from $w = 6$ data vectors with a single one in position i for data vector i. In base five the AND operation is equivalent to multiplication modulo five and the exclusive or (XOR) operation is equivalent to addition modulo five. For a data vector [100000], the parity bits $P_{1,1}, P_{2,1}, P_{3,1}, P_{1,2}, P_{2,2}, P_{3,2}$ are calculated as follows:

1 0 0 0 0 0

1 0 1 2 1

———

1+0+0+0+0 = 1 MODULO 5 = $P_{1,1} = 1$

1 0 0 0 0 0

1 3 1 2 1

———

1+0+0+0+0 = 1 MODULO 5 = $P_{2,1} = 1$

1 0 0 0 0 0

2 1 2 1 1

———

2+0+0+0+0 = 2 MODULO 5 = $P_{3,1} = 2$

Shift C_1 by $k = 1$ to get next parity bit

0 0 0 0 0

1 0 1 2 1

———

0+0+0+0+0 = 0 MODULO 5 = $P_{1,2} = 0$

Shift C_2 by $k = 1$ to get next parity bit

0 0 0 0 0

1 3 1 2 1

———

0+0+0+0+0 = 0 MODULO 5 = $P_{2,2} = 0$

Shift C_3 by $k = 1$ to get next parity bit

0 0 0 0 0

2 1 2 1 1

———

2+0+0+0+0 = 0 MODULO 5 = $P_{3,2} = 0$

From the example above, 112000 are the corresponding parity bits for data sequence 100000. Following the same procedure we obtain Table 3.7 as the resulting reduced encoding table for the base five convolutional encoder described above.

Encoding proceeds similar to binary table based encoding. The only difference is that the basis data vectors in the reduced encoding table can be multiplied by all elements in the

Table 3.7: Reduced Encoding Table for (3,1,4) Base 5 Convolutional Code

Data Bits	Parity Bits
100000	112000
010000	031112
001000	112031
000100	221112
000010	111221
000001	000111

field of five. For example, given the reduced encoding table in Table 3.7, the corresponding parity bit for the data vector (3 0 2 0 0 2) can be found by multiplying and summing (all modulo five) the basis data vectors and their corresponding parity vectors as follows:

3*(1 0 0 0 0 0)

0*(0 1 0 0 0 0)

2*(0 0 1 0 0 0)

0*(0 0 0 1 0 0)

0*(0 0 0 0 1 0)

2*(0 0 0 0 0 1)

———————

3 0 2 0 0 2 = Data

Corresponding parity:

3*(1 1 2 0 0 0)

0*(0 3 1 1 1 2)

2*(1 1 2 0 3 1)

0*(2 2 1 1 1 2)

Table 3.8: Reduced Decoding Table for (3,1,4) Base 5 Convolutional Code

Parity Bits	Data Bits
100000	430312
010000	320024
001000	200112
000100	103431
000010	002240
000001	400430

0*(1 1 1 2 2 1)

2*(0 0 0 1 1 1)

———————

0 0 0 2 3 4 = Parity

The above can be written using matrices, where D is the data matrix of basis data vectors and P_{data} is the corresponding parity vectors for the basis vectors. Let u represent the w-bit input vector, then the corresponding parity vector, v, is calculated as follows:

$$v = u * P_{data} \tag{3.10}$$

3.3.2 Base Five Table-Based Decoding

The decoding methods in base five are the same as in binary. To find the reduced decoding table, P, and its corresponding data vectors D_{parity}, simply calculate the inverse of P_{data}:

$$D_{parity} = P_{data}^{-1} \tag{3.11}$$

For the code in Equation 3.9, the reduced decoding table is shown in Table 3.8 Similar to encoding, decoding can be performed using matrices. To find the data vector, u, corresponding to the received parity vector $r = (0\ \ 0\ \ 0\ \ 2\ \ 3\ \ 4)$, we use the following equation:

$$u \ = \ r * D^{parity} \tag{3.12}$$

This becomes

$$u = \begin{bmatrix} 0 & 0 & 0 & 2 & 3 & 4 \end{bmatrix} \begin{bmatrix} 4 & 3 & 0 & 3 & 1 & 2 \\ 3 & 2 & 0 & 0 & 2 & 4 \\ 2 & 0 & 0 & 1 & 1 & 2 \\ 1 & 0 & 3 & 4 & 3 & 1 \\ 0 & 0 & 2 & 2 & 4 & 0 \\ 4 & 0 & 0 & 4 & 3 & 0 \end{bmatrix} \tag{3.13}$$

and

$$u = [3\ \ 0\ \ 2\ \ 0\ \ 0\ \ 2]$$

3.3.3 Formation of the Gmask

As in the binary case, overlapping bits of shifted error-less parity streams should result in all zero difference vectors. For the code in Equation 3.9, the gmask length will equal $w+n = 9$ and there will be $n-k = 2$ different gmasks. The procedure for forming the base five gmask is very similar to the binary case, with a few notable exceptions.

The following procedure describes how to generate the base five g-mask, given a reduced decoding table.

- Consider parity streams with single bit errors in each position of the n bit parity block $[P_1\ \ P_2\ \ \cdots\ \ P_n]$. The error bit value is equal to one. These are the basis error vectors. All other one bit error patterns can be formed by multiplication (modulo five) of the basis error vectors with scalars in the field of five.

- Decode the error parity stream w bits at a time. For the w-bit parity sequence, p_i in the parity stream, find the corresponding w-bit data sequence d_i.
- Shift n-bits in the parity stream and find the w-bit data sequence,d_{i+1} that corresponds to the next w-bit parity sequence, p_{i+1}. As in binary table-based decoding, the last $w-k$ bits of p_i will overlap the first $w-k$ bits of p_{i+1}. Calculate the bitwise difference vector, $dvec_i$ by subtracting the overlapping bits in p_{i+1} from the overlapping bits in p_i. In binary, subtraction and addition are synonymous, but this is not true for base five. Table 3.6 shows subtraction base five.

  In binary there were up to 2^{n-k} difference vectors, but as one would expect, in base five there are at most 5^{n-k} possible difference vectors.
- Each difference vector (pattern) is assigned an $n-k$ bit syndrome value between 0 and 5^{n-k} as follows:
  - Assign the first non-zero difference pattern the $n-k$ base five bit value of $5^0 = 1$.
  - Assign the second non-zero unique difference pattern the $n-k$ base five bit value of $5^1 = 5$. A non-zero unique difference pattern is a pattern that is not a multiple of any other unique difference pattern.
  - Continue assigning syndrome values to all difference patterns based on the following rule. If difference pattern is the N^{th} non-zero unique difference pattern then its syndrome equals the $n-k$ base five bit value of 5^{N-1}. Else, the difference pattern must be a scalar multiple, M, of a unique difference pattern $dvec_i$(basis difference pattern); its syndrome pattern is equal to $M * svec_i$, where $svec_i$ is the basis syndrome for $dvec_i$.

An example of syndrome assignment and gmask formation follows.

- Once the $n-k$ bit syndromes have been found for each error parity stream, $n-k$ gmasks are formed from the syndrome values. The g-mask is formed by interleaving the individual n syndrome vector bits generated for each shift of the corrupted parity stream. For an $n=3$, $k=1$ code, the syndrome vectors are made of two bits: $svec = [s1 \quad s2]$. Gmask1 is formed by interleaving the $ns1$ syndrome bits and gmask2 is formed by interleaving the $ns2$ syndrome bits. See example below.

The following illustrates how to form the base five gmask for an $n=3, k=1, m=4$ code given the reduced encoding and decoding table. The gmask is for the code in Equation 3.9.

```
-----------------------------------------------------------------
How to form gMask in base 5 Example: Rate=1/3 (n=3, k=1), L=5,
                                     w=n(L-k)/(n-k)=3(5-1)/(3-1)=6
                                     gmask length=w+n=9

CODE: C1=1 0 1 2 1  C2=1 3 1 2 1    C3=2 1 2 1 1


Reduced Encoding Table:
    Data    |    Parity
-------------------------
1 0 0 0 0 0 | 1 1 2 0 0 0

0 1 0 0 0 0 | 0 3 1 1 1 2

0 0 1 0 0 0 | 1 1 2 0 3 1

0 0 0 1 0 0 | 2 2 1 1 1 2

0 0 0 0 1 0 | 1 1 1 2 2 1

0 0 0 0 0 1 | 0 0 0 1 1 1


Reduced Decoding Table:
    Parity  |    Data
-------------------------
1 0 0 0 0 0 | 4 3 0 3 1 2
```

```
0 1 0 0 0 0 | 3 2 0 0 2 4

0 0 1 0 0 0 | 2 0 0 1 1 2

0 0 0 1 0 0 | 1 0 3 4 3 1

0 0 0 0 1 0 | 0 0 2 2 4 0

0 0 0 0 0 1 | 4 0 0 4 3 0
```

```
To produce gmask, introduce error in parity bits of P1, P2, and P3
then assign syndromes to the resulting difference patterns.  There
will be n-k=3-1=2 basis syndrome values: (01,10).  From these all
other syndrome values will be multiples.  (The difference vectors
are numbered d1 through d9, with the basis vectors being
referenced as d1 and d2).

P1
    PARITY             DATA      DIFFERENCE  | SYNDROME
000000100000000.......................................
000000----------------000000                 s1 s2
   000100-------------- 103431    ==> 40212  | 0 1 <--1st basis=d1
      100000-----------  430312   ==> 10400  | 1 0 <--2nd basis=d2
         000000--------   000000  ==> 30312  | 4 1 <--d1+4(d2)
--------------------------------------------------------------
P2
    PARITY             DATA      DIFFERENCE  | SYNDROME
000000010000000.......................................
000000----------------000000                 s1 s2
   000010-------------- 002240    ==> 00331  | 3 3 <--3(d1)+3(d2)
      010000-----------  320024   ==> 20243  | 1 4 <--4(d1)+d2
         000000--------   000000  ==> 20024  | 4 2 <--2(d1)+4(d2)
--------------------------------------------------------------
P3
    PARITY             DATA      DIFFERENCE  | SYNDROME
000000001000000.......................................
000000----------------000000                 s1 s2
   000001-------------- 400430    ==> 10012  | 2 1 <--d1+2(d2)
      001000-----------  200112   ==> 30424  | 0 2 <--2(d1)+0(d2)
         000000--------   000000  ==> 00112  | 1 1 <--d1+d2
--------------------------------------------------------------

*The multiplication in the parentheses means that the syndrome
value is produced by multiplying a basis syndrome value by a
scaler.  The corresponding difference value is also a product of
the same scaler and the difference vector of the associated
syndrome value.
```

```
Separate syndromes by shifts:

   S1 S2        S1 S2        S1 S2
   -- --        -- --        -- --
P1 0  1      P2 3  3      P3 2  1 <-LINE1: shift 1 of parity
   1  0         1  4         0  2 <-LINE2: shift 2 of parity
   4  1         4  2         1  1 <-LINE3: shift 3 of parity

Gmask1 is made up of the interwoven syndrome one bits, s1, from
each parity position (P1, P2, P3) for each shift.  The first shift
is the last set of three syndrome bits:

gMask1 = 441 110 032

Gmask2 is formed in the same manner as gmask1 except using
syndrome two, s2, bits:

gMask2 = 121 042 131

-----------------------------------------------------------------
```

There will be $n - k$ gmasks for an n, k code. Once the gmask has been constructed, it can be used to calculate the syndrome vector for the parity stream. To calculate the syndrome vector using the g-mask:

- The gmask is ANDed with the first $w + n$ parity bits.
- The result is exclusive-ORed to produce a syndrome value.
- The received parity stream is shifted by n bits.
- The process is repeated until all syndrome values of the syndrome vector are produced. Each shift results in one syndrome value.

```
-----------------------------------------------------------------

EXAMPLE SYNDROME CALCULATION FOR ERRORLESS PARITY STREAM

Test the gmask for an errorless parity stream to see if it
produces the all zero syndromes.

TEST
----
```

```
Data:   000 000 100 000

Parity: 000 000 000 111 221 112 031 112 000 000 000

(zero padded front and back)

Generate syndrome using gmask1 and gmask2 by finding the modulo
five dot product between the w+n=9 bit sub-vector of the parity
sequence and each nine bit gmask. Then shift the parity sequence
by n=3 and repeat.  For each shift of the parity sequence, n-k=2
syndrome bits are produced.

                     PARITY          gMASK         SYNDROME
Shift1(gmask1): <000 000 000> * <441 110 032>   =    0

Shift1(gmask2): <000 000 000> * <121 042 131>   =    0
-----------------------------------------------------------
Shift2(gmask1): <000 000 111> * <441 110 032>   =    0

Shift2(gmask2): <000 000 111> * <121 042 131>   =    0
-----------------------------------------------------------
Shift3(gmask1): <000 111 221> * <441 110 032>   =    0

Shift3(gmask2): <000 111 221> * <121 042 131>   =    0
-----------------------------------------------------------
Shift4(gmask1): <111 221 112> * <441 110 032>   =    0

Shift4(gmask2): <111 221 112> * <121 042 131>   =    0
-----------------------------------------------------------
Shift5(gmask1): <221 112 031> * <441 110 032>   =    0

Shift5(gmask2): <221 112 031> * <121 042 131>   =    0
-----------------------------------------------------------
Shift6(gmask1): <112 031 112> * <441 110 032>   =    0

Shift6(gmask2): <112 031 112> * <121 042 131>   =    0
-----------------------------------------------------------
Shift7(gmask1): <031 112 000> * <441 110 032>   =    0

Shift7(gmask2): <031 112 000> * <121 042 131>   =    0
-----------------------------------------------------------
Shift8(gmask1): <112 000 000> * <441 110 032>   =    0

Shift8(gmask2): <112 000 000> * <121 042 131>   =    0
-----------------------------------------------------------
Shift9(gmask1): <000 000 000> * <441 110 032>   =    0

Shift9(gmask2): <000 000 000> * <121 042 131>   =    0
-----------------------------------------------------------
```

Hence, the syndrome vector for the above errorless example would be:

$$S - vector(g1, g2) = [00\ \ 00\ \ 00\ \ 00\ \ 00\ \ 00\ \ 00\ \ 00\ \ 00]$$

The all zero syndrome vector indicates a parity sequence produced by the code which generated the gmask(s).

In the following chapter, base five and binary table-based coding is used to search for "good" gmasks for prokaryotic initiation regions.

Chapter 4

Table-Based Codes for Prokaryotic Translation Initiation Systems

The error control codes that accurately describe translation initiation most likely exist in a multi-dimensional coding space. This work develops and evaluates separate coding models for initiation in two dimensions: horizontal and vertical codes. This chapter summarizes current methods for finding "good" convolutional codes, presents an overview of genetic algorithms (GAs) and discusses the use of genetic algorithms for finding "good" table-based codes for prokaryotic translation initiation. Results are presented and discussed in the final sections of this chapter. The concepts presented in the second section of this chapter draw extensively on D. A. Coley's discussion of genetic algorithms [7].

4.1 Finding Optimal Convolutional Codes

The majority of methods for constructing optimal convolutional codes rely on computer search techniques [60]. Searches attempt to discover codes which maximize a specified coding distance measure such as the Hamming distance, or the free Euclidean distance for trellis coded modulation (TCM) codes [60]. Code construction techniques which use exhaustive or code search algorithms can be time consuming and possibly fail to locate optimal codes [61, 62, 60]. The ability to locate the best codes for a given set of parameters is dependent on the efficiency of the algorithms used. Algorithm efficiency is particularly important when searching for codes with long memory lengths [60]. As the code length increases, the search space increases at an exponential rate [63].

Some researchers have explored the use of genetic algorithms to search for optimal convolutional codes. Barnes [63] used a genetic algorithm to locate near-optimal codes for half-rate table-based, convolutional codes. Kotrys and Remelein used genetic algorithms to find good trellis coded modulation (TCM) codes [60]. The GAs were able to locate codes near known optima, equivalent to currently known optimal codes. For longer memory lengths (for which no known optima exists), the GAs were able to locate optimal codes [63, 60].

4.2 Genetic Algorithms [7]

Genetic algorithms are numerical optimization techniques based on a generalized view of the theory of evolution, natural selection, and genetics. Invented in the 1960s by John Holland, GAs have been effectively applied to a wide range of optimization problems of varying size and complexity. Application areas include image processing, three dimensional protein structure predictions, time series analysis, and many other fields. Genetic algorithms perform especially well in problems where the solution space is filled with numerous local optima. The optimal table-based code for the translation initiation system probably resides in such a solution space.

4.2.1 Overview of Genetic Algorithms

An optimization algorithm searches a, possibly infinite, list of viable solutions for the solution(s) that best solves the posed question. The list of possible solutions is called the search space. The fitness of a solution is a measure of how successfully the candidate solution solves the problem. To preserve the concept of a search space for multi-dimensional problems, it must be possible to evaluate the fitness of individual solutions and define a measure of the distance between solutions.

The highest peak (or trough, depending on the problem) in fitness value is referred to as the global optimum (maximum or minimum, depending on the problem). Smaller peaks (or troughs) in the search space are called local optima (maximum or minimum). The goal of most optimization problems is to locate the global optimum. Depending on the search objective, location of local optima or location of solutions that are above (or below) a predetermined threshold may be sufficient or preferred. This work searches for an optimal code for each leader sequence in the training set.

Traditional algorithms for locating the optimal solution in a given search space include enumerative searches and direct searches. Enumerative searches estimate the value of the unknown parameter(s) by solving the given problem for a large set of possible solutions. They select the best solution based on minimization of a cost or objective function. Enumerative searches are suitable for problems with a small number of parameters and a rapid algorithm for calculating the objective function. For problems with large search spaces or with computationally intensive objective functions, enumerative searches are not efficient.

Direct searches begin with two possible solutions and based on the value of the objective function at those solution points, the next point is selected at a distance δ from the current point. The incremental step size, δ, used to compute the next solution point can be dynamically adjusted. The drawbacks of direct search algorithms include: the algorithm can not be universally applied and the final solution is dependent on the initial starting point. Direct search algorithms can become trapped in a local optima and fail to locate the optimal solution. In addition, there is a lower confidence associated with the final answer since it is dependent on the algorithm's starting point. Therefore, in complex search spaces with multiple local optima, the direct search algorithm is impractical.

Although simulated annealing algorithms and random searches have been used to find optimal solutions in complex search spaces, random searches guided by concepts that parallel evolution and genetics, have been the most effective of these types of algorithms. Genetic algorithms fall into the random search category.

4.2.2 Components of a Genetic Algorithm

There are four elements which constitute a typical genetic algorithm:

- A population of possible solutions from the problem's solution space. Each possible solution is called an individual.
- A method for evaluating the *fitness* of the individual. *Fitness* is a measure of how well the proposed solution or individual solves the problem being investigated.
- An approach for combining the better, or more fit, individuals to form new solution populations with higher average fitness values.
- A mutation method for preserving diversity within the population of individuals.

Genetic algorithms are initialized with a population of usually random possible solutions from the solution search space. The typical size of the initial population ranges anywhere from twenty to one thousand individuals; this number can be smaller or greater depending on the problem. Using three main genetic operators, selection, crossover, and mutation, the initial population "evolves," over a set number of iterations or generations, towards convergence to the global optimal.

Typically, each individual solution is represented as a binary vector called a chromosome. The genetic operators, operate on the chromosome. The binary chromosome is then converted to the appropriate representation for the given application and the individual solution is evaluated and assigned a fitness value. Increasing numbers of GAs are using real-valued encoding instead of binary representations for chromosomes. In this work, binary vectors are used to represent chromosomes of individuals in the population.

The following outlines the basic steps for a typical GA:

1. Initialize - Set all probability parameters (including crossover and mutation probability thresholds).
2. Generation=1
3. Create Initial Population - Construct a random population of binary strings (chromosomes).
4. Find Unknowns - Convert the binary chromosomes into the application specific parameters (integers, real numbers, etc.).
5. Assign Fitness - Calculate the fitness of each individual in the population based on some optimization criterion.
6. `For Generation = 2 to MAX_NUMBER_OF_GENERATIONS`
    - *Loop over current population and select pairs of mates*
    - `For New_Individual = 1 to POPULATION_SIZE/2`
        - o Select Parent One
        - o Select Parent Two
        - o Perform Crossover - Produce children (two at a time) by crossing the binary chromosomes of selected parents.
        - o Put new individuals into a temporary population
        - o `NEXT New_Individual`

- Mutate - Mutate each individual in the temporary population.
- Replace - Replace the old (current) population with the new population (contained in temporary population).
- Find Unknowns - Convert the binary chromosomes (in current population) into the application specific parameters (integers, real numbers, etc.).
- Assign Fitness - Calculate the fitness of each individual in the current population based on some optimization criterion.
- `NEXT Generation`

7. END GA

4.2.3 Genetic Operators

As previously mentioned, there are three main genetic operators used in GAs: selection, crossover, and mutation. Each operator helps move the population towards the optimal solution.

Selection

The selection operator applies pressure to the population similar to natural selection in biological systems. During selection, individuals with high fitness values are selected over low fitness individuals. Individuals with high fitness values create the new breeding population. Hence, individuals with high fitness values (good solutions) have a greater than average chance of passing on their information to the next generation. Some of the selection methods used by GAs include:

- Select the top fifty percent of individuals (based on fitness values) for reproduction and discard the rest. This selection technique does not distinguish good individuals from very good individuals. Another drawback is that low fitness individuals are completely annihilated. This reduces the overall genetic diversity of the population.

- Fitness-proportional selection, also referred to as roulette wheel selection, distinguishes between good and very good solutions. In fitness-proportional selection, an individual's probability of selection is proportional to the fitness of the individual. Similar to a roulette wheel, the higher the fitness value of an individual, the larger the arc associated with the individual on a theoretical roulette wheel. The circumference, CIRC, of the wheel is the sum of all fitness values. The spinning of the wheel is simulated by assigning a ball, B, a random number between zero and CIRC. The selection rule is:

$$Sum_k = \sum_{i=0}^{k} f_i$$

  IF $Sum_k > B$ AND $Sum_{k-1} < B$ THEN select individual k

  In the summation equation, f_i represents the fitness value for the *ith* individual.

- Target sampling rates or *tsr* values can be used to select parents from the current population [63]. Each individual is assigned a fitness-associated target sampling rate. An individual's tsr value indicates the number of times they can be selected for mating. For example, an individual with $tsr = 3$ can be selected as a parent three times while a less fit individual with $tsr = 1$ can only be a parent once. Target sampling rates are assigned as follows:

$$tsr(i) = \frac{f_i}{f_{avg}} \tag{4.1}$$

  where f_{avg} is the average fitness for the population. Once all *tsr* values have been assigned, selection proceeds as follows:

  1. Select a number, I, between 1 and POPULATION_SIZE.

2. If $tsr(I) > 0$ Then
    - Select individual I
    - Update tsr: $tsr(I) = tsr(I) - 1$

Even though the selection of an individual is not fitness-proportional, the ability to reproduce is increased or decreased by an individuals fitness.

Selection methods such as fitness-proportional selection can result in pre-mature convergence of the genetic algorithm to a local optima. Applying fitness scaling to the individual fitness values can prevent highly fit individuals from flooding the population and causing pre-mature convergence.

Fitness Scaling

Linear fitness scaling translates individual fitness values around the population average. Scaling allows the most fit individuals to be selected a constant number, C_m, of times as much as individuals with average fitness. Raw fitness values are translated into scaled fitness values using the following linear scaling equations:

$$f_i^s(g) = a(g)f_i(g) + b(g) \tag{4.2}$$

where $f_i(g)$ is the unscaled fitness value for the ith individual, $f_i^s(g)$ is the scaled fitness value for the ith individual, and g is the generation counter. The coefficients $a(g)$ and $b(g)$ are defined as:

$$a(g) = \frac{(C_m - 1)f_{avg}(g)}{f_{max}(g) - f_{avg}(g)} \tag{4.3}$$

where $f_{avg}(g)$ is the average fitness for generation g and $f_{max}(g)$ is the max fitness value for generation g.

$$b(g) = (1 - a(g))f_{avg}(g) \tag{4.4}$$

During each selection iteration, the selection algorithm is applied twice to select a pair of parents to mate using the crossover operator.

Crossover

The crossover operator is a recombination technique that allows individuals in the current population to exchange "genetic" information, similar to the exchange of genetic information by biological organism during sexual reproduction. In a GA, crossover occurs with probability P_C. Typical values for P_C range from 0.4 to 0.9. There are several crossover methods. The main techniques are described below.

Single Point Crossover: In single point crossover, the pair of individuals or parents, $P1$ and $P2$, selected using the selection operator are crossed at a single position in their binary chromosomes. Single point crossover proceeds as follows:

1. Generate a random number, p_c, between 0 and 1.
2. If $p_c \leq P_C$ then proceed with crossover (goto next step); else set
$$child1 = P1$$
$$child2 = P2$$
3. Randomly select a position, POS, in the chromosome; POS will be between 1 and $LENGTH\_CHROMOSOME - 1$.
4. Swap the information to the right of POS to produce child1 and child2.

For example, assume the following parents, $P1$ and $P2$, are selected:

$$P1 = 1\ \ 1\ \ 0\ \ 0\ \ 1\ \ 0$$

$$P2 = 0\ \ 1\ \ 0\ \ 1\ \ 0\ \ 1$$

If the crossover point is randomly selected as,

$$POS = 2$$

Then *child1* is composed of the first two bits of $P1$ and the last four bits of $P2$ while *child2* contains the first two bits of $P2$ and the last four bits of $P1$:

$$child1 = 1 \;\; 1 \;\; 0 \;\; 1 \;\; 0 \;\; 1$$

$$child2 = 0 \;\; 1 \;\; 0 \;\; 0 \;\; 1 \;\; 0$$

In single point crossover, if the parents are identical in the region to the right of the crossover position, the children will be identical to the parents.

Multipoint Crossover: Multipoint crossover is similar to single point crossover except multipoint crossover allows the selection of multiple crossover points. Given a GA that employs two point crossover, using the same parents from the previous example, if crossover point $POS_1 = 1$ and the second crossover point $POS_2 = 4$ the resulting children would be:

$$child1 = 1 \;\; 1 \;\; 0 \;\; 1 \;\; 1 \;\; 0$$

$$child2 = 0 \;\; 1 \;\; 0 \;\; 0 \;\; 0 \;\; 1$$

Uniform Crossover: Taking multipoint crossover to its limit, uniform crossover forces bits to be exchanged at every point or locus. This can be disruptive and negatively affect the GA. But parameterized crossover, a form of uniform crossover, applies a probability to each locus to determine whether crossover will occur at that locus[7, 63] The probability of crossover occurring at a locus can range from 0.5 to 0.8. *Note, this is not P_C.*

As an example, given the above parents, assume the probability of crossover at each locus is represented by the following vector of probabilities:

$$P_{locus} = 0.51 \;\; 0.7 \;\; 0.22 \;\; 0.02 \;\; 0.31 \;\; 0.1$$

A crossover mask, the method used by Barnes in [63], can be generated by putting a 0 where the probability is less than the threshold and a 1 where the probability is greater than or equal to the threshold. A 0 indicates a non-crossover locus and a 1 indicates a crossover locus. For a locus crossover threshold of 0.50, the above P_{locus} vector results in the following crossover mask:

$$CrossoverMask = 1 \;\; 1 \;\; 0 \;\; 0 \;\; 0 \;\; 0$$

This particular crossover mask produces the following children given the parents from the single-point crossover example:

$$child1 = 0 \;\; 1 \;\; 0 \;\; 0 \;\; 1 \;\; 0$$

$$child2 = 1 \;\; 1 \;\; 0 \;\; 1 \;\; 0 \;\; 1$$

The crossover operator enables exploration of new regions of the search space.

Mutation

In addition to crossover, mutation helps the GA further explore the search space and possibly frees the GA from local optima solutions. Mutation randomly flips the binary digits in an individuals binary chromosome. Mutation is used sparingly. The probability of mutation (for each binary digit) is determined by P_M which is generally of the order 0.001. The probability of mutation is application dependent. Possible values include:

$$P_M \approx \frac{1}{L_{chromosome}}$$

or

$$P_M \approx \frac{1}{N\sqrt{L_{chromosome}}}$$

where $L_{chromosome}$ is the length of the binary chromosome vector. Given P_M, the mutation rule is:

- Mutation Rule: For each bit in every individual chromosome, randomly select a number, p, between zero and one. If $p < P_M$ then flip the binary bit (i.e. if current bit is a zero, change it to a one and vice versa); else leave the current binary bit value.

Mutation is the last genetic operator applied to the temporary population prior to fitness assignment. After fitness evaluation, the genetic algorithm can use *elitism* to increase the fitness of the current population.

Elitism

During the run of a genetic algorithm there is a chance that the most fit individual from the previous generation may not be selected for reproduction. It is also possible that all the individuals in the current generation are less fit than the most fit individual (the elite member) from the previous generation. To guarantee that the elite member of the current generation is as fit or more fit than the elite member of the previous generation, a genetic algorithm can employ elitism. Elitism is carried out as follows:

1. If the current generation's elite member is less fit than the previous generation's elite member, proceed to step 2.
2. Randomly select a number, I, between 1 and POPULATION_SIZE.
3. Replace individual I with the elite individual from the previous generation.
4. The elite individual from the previous generation is now the current generation's elite member.

Genetic operators and techniques are selected based on the type of problem the GA is attempting to solve. This work uses a genetic algorithm and the associated genetic operators to search for optimal table-based codes for leader regions of prokaryotic organisms.

Table 4.1: Mapping of RNA bases to the finite field of five.

RNA Base	Field 5 Representation
Inosine	0
Adenine	1
Guanine	2
Cytosine	3
Uracil	4

4.3 Genetic Algorithms for Table-Based Translation Initiation Codes

As discussed in preceding chapters, previous work [11] demonstrated the plausibility of using coding theory concepts to describe the translation initiation region. It also demonstrated the feasibility of mapping nucleic acid bases to the field of five representation based on biological characteristics. The RNA alphabet consists of inosine (found in transfer RNA), adenine, guanine, cytosine, uracil (thymine replaces uracil in DNA) or I, A, G, C, U (or T in DNA), respectively. The corresponding field of five is composed of the symbols 0, 1, 2, 3, 4, and the operations addition modulo five and multiplication modulo five. The RNA bases for the field of five are defined in Table 4.1. The corresponding multiplication and addition operations are defined in Chapter 3, Section 3.3. Given these RNA to base five mappings, we can use the base five tables from Chapter 3 to implement all coding based operations.

The basic premise of this work is that the leader region of prokaryotic messenger RNAs can be viewed as noisy encoded sequences. Previous research demonstrated that both a sliding block code and a convolutional code can be used to describe the translation initiation region in a general sense. Present work aims to develop a set of convolutional coding models that describe individual leader sequences in a specific sense. Unlike block code design which has proven construction methods [54, 20], "good" convolutional codes are designed using various search techniques [61, 62, 60] and, in recent years, genetic algorithms [63, 60]. The definition of a "good" convolutional code is usually based on the memory length, error detecting, and error correcting capabilities of the code. For genetic convolutional codes, those same parameters will affect the code, but for this work the most important feature of the code is how well it distinguishes errors from non-errors, non-ribosome binding sites from ribosome binding sites.

Based on the error detecting nature of codes, channel codes can be generally defined as pattern recognition systems as discussed by Savchenko in [64]. The codewords produced by the code are the patterns the system needs to recognize. In an ideal case, a "good" code will recognize the patterns the system wants with a probability of one and all non-system patterns with a probability of zero. (In reality, channel codes go beyond a typical pattern recognizer. They recognize and correct patterns that are close to the system patterns.) This work defines a "good" code based on how well the code recognizes the "patterns" or RNA bases that form the leader region. Based on this requirement, and given the large search space for genetic codes, it proved efficient to use genetic algorithms to search for good genetic convolutional codes.

The objective of the genetic algorithm search is to find a convolutional code (a set of generators) with the highest probability of producing each individual leader sequence in the *E. coli* training set. Location of a set of convolutional codes which recognize individual leader sequences will demonstrate that in a specific sense, the ribosome can be modeled as a table-based convolutional decoder. Also, the design of genetic convolutional codes using genetic algorithms is the first step towards designing efficient translation initiation sites for transgenic protein production.

4.3.1 Methodology

Assuming that the ribosome decodes the mRNA leader region similar to a table-based decoder, a genetic algorithm is used to search for table-based convolutional codes whose gmasks recognize individual ribosomal binding site sequences similar to the conjectured functional behavior of the 16S ribosomal RNA. As discussed, when the syndrome is zero then there should be no transmission errors in the parity sequence unless the error-rate exceeds the power of the code. Specifically, a syndrome value of zero indicates that no errors occurred, within the codes power or ability to detect t errors. Given a candidate code, the gmask of the code is used to determine the fitness of the code. The GAs search space includes all possible (n, k, m) convolutional codes. Performing a systematic fitness evaluation of all possible codes for each sequence takes a long time. For example, searching for an optimal $(n = 3, k = 1, m = 4)$ code would take a little over two days per sequence and over a year for the given training set (using a 600 MHz Microsoft Windows 2000 machine)! Using a GA allowed for quick exploration of the search space and led to a relatively shorter search time for locating the optimal solution.

As previously stated, a genetic algorithm is composed of a population of potential solutions, a method for evaluating the fitness of each potential solution or individual, and genetic operators to recombine more fit individuals to form new and diverse potential solutions. Each aspect of the GA was defined based on the objective: to locate a $(n = 3, k = 1, m = 4)$, $L = 5$, convolutional code which has the greatest probability of producing the messenger RNA parity sequence for the *E. coli* leader sequences evaluated. The $(n = 3, k = 1, m = 4)$, $L = 5$, convolutional code was used to verify the proposed code construction method; it is most likely not the best code model. It is believed that longer code models (and possibly models with different code rates) will result in better translation initiation models than those constructed using the $(n = 3, k = 1, m = 4)$, $L = 5$, convolutional code. Also, the GA search space grows exponentially as the code length increases and as n increases. Executing the code construction methodology for longer codes is not easily realizable using the current system (a 600 MHz Microsoft Windows 2000 machine).

Individuals

For the $(3, 1, 4)$ convolutional code, there were three generators of length $L = m + 1$ or five. In a multi-parameter genetic algorithm (which this was, since there were three generators with five coefficients in each) each solution can be represented as a concatenation of all the parameters. The i_{th} parameter was represented by B_{param_i} bits. So, each parameter could be represented with different numbers of bits.

In the $(3, 1, 4)$ genetic coding GA, there were a total of fifteen parameters. Given the base five representation of RNA, the parameter values could range from zero to four. But, since inosine (I=0) is not found in mRNA leader sequences and to reduce the search space, the GA only considered codes with parameters ranging from one to four (or A, G, C, and U). Although the code parameters did not contain zeros, the associated gmask, which was used to evaluate the fitness of the code, did contain zeros. This is evident from the gmask construction procedures presented in Chapter 3. With four possible values for each coefficient in the three generators, each coefficient was represented using $B_{param_i} = B_{param} = 2$ bits for all i. There were a total of $n * L * B_{param} = 30$ bits in the binary chromosome representation of each code or individual. The search space contained a total of 1,073,741,824 candidate codes. One can appreciate how quickly the problem grows for longer codes and for larger values of n.

Although, the gmask was used to evaluate fitness, the objective was to find the code which best described the mRNA leader sequences. Since the number of possible gmasks, for the $(3, 1, 4)$ base five convolutional code was 1,953,125 it may seem more logical to search for the optimal gmask. The drawbacks of starting with the gmask and working backward to a candidate code were: different codes could produce the same gmask, some code candidates were not valid (they were not invertible) and some gmasks may be associated with the invalid codes. Finding a code had more value since the code could serve as the foundation for constructing heuristics that govern the design of viable ribosome binding sites and for developing algorithms that could potentially “correct” inefficient sites transforming them into highly efficient translation initiation sites.

Each individual in the population contained eight elements used by the genetic operators of the genetic algorithm. An individual was represented by the *individual* structure defined as (C code description):

```
struct individual {
    /*Define contents of a member of the population;
    stores key info for each individual*/
    int binvec[2*N*L];      //binary rep of individual (code)
    int decTag;             //decimal digit rep of individual
    int code[N][L];         //base 5 rep of individual
    int gmask[N-K][gLENGTH];//gmask associated with code
    double sdist[nGSHIFTS]; //average syndrome dist vector
    double fitness;         //fitness of individual
    double sfitness;        //scaled fitness of individual
    int tsr;                //target sampling rate
};
```

The elements that constitute an individual illustrate the functions, procedures, and information used by the GA to locate global and/or local optima code models for translation initiation. Further description of each element follows.

- Binary Chromosome - The binary representation of the code. Crossover and mutation procedures used the binary chromosome to produce new and diverse individuals for the GA population. The initial population of individuals were constructed by generating $N = POPULATION\_SIZE$ binary chromosomes randomly. The following generates binary chromosomes:

```
/**MKINDIVIDUAL makes a binary vector of length 2*N*L by
    generating a random 0 or 1 for each position.
    VARS:   Output-indiv[2*N*L]=binary vector of generated bits
***/
int mkindividual(int indiv[]) {
    int i;
    double prand;   //percent value of random integer

    for (i=0; i<lengthIndiv; i++)
        {
        indiv[i]=random(2);
        }   //end i

    return(0);

};  // mkindividual(int indiv[])
```

In the above, $lengthIndiv = B_{param}$, which is thirty. Hence, the binary vector

$$111000001011100011000000111100$$

is a possible binary chromosome.

- Decimal Tag - The base ten integer representation of the code. Each binary chromosome was converted to its decimal equivalent. For an (n, k, m) $(L = m+1)$ code, the decimal tag values ranged from 0 to $2^{2nL} - 1$. The decimal tag was a shorthand representation of the individual and was not used by any GA operators.
- Candidate Convolutional Code - After creating the initial population of binary chromosomes, the next step was to convert the binary chromosomes into the coding parameters or the coefficients of the code. Table 4.2 was used to convert every two bits in the chromosome to RNA bases which were the coding parameters. The example binary chromosome (grouped every two bits)

$$[11\ \ 10\ \ 00\ \ 00\ \ 10\ \ 11\ \ 10\ \ 00\ \ 11\ \ 00\ \ 00\ \ 00\ \ 11\ \ 11\ \ 00]$$

Table 4.2: Conversion of Binary Chromosome to Application Specific Parameters

Binary	Code Parameters
00	A (1)
01	G (2)
10	C (3)
11	U (4)

converts to the RNA code parameters

$$[U \;\; C \;\; A \;\; A \;\; C \;\; U \;\; C \;\; A \;\; U \;\; A \;\; A \;\; A \;\; U \;\; U \;\; A]$$

Taking every $L = 5$ bases to represent the generator coefficients for C_1 to C_n, where $n = 3$, we get the following genetic convolutional code:

$$C_1(d) = [U \;\; C \;\; A \;\; A \;\; C] = Ud + Cd^{-1} + Ad^{-2} + Ad^{-3} + Cd^{-4}$$

$$C_2(d) = [U \;\; C \;\; A \;\; U \;\; A] = Ud + Cd^{-1} + Ad^{-2} + Ud^{-3} + Ad^{-4}$$

$$C_3(d) = [A \;\; A \;\; U \;\; U \;\; A] = Ad + Ad^{-1} + Ud^{-2} + Ud^{-3} + Ad^{-4}$$

The base five convolutional code is then

$$C_1(d) = [4 \;\; 3 \;\; 1 \;\; 1 \;\; 3] = 4d + 3d^{-1} + d^{-2} + d^{-3} + 3d^{-4}$$

$$C_2(d) = [4 \;\; 3 \;\; 1 \;\; 4 \;\; 1] = 4d + 3d^{-1} + d^{-2} + 4d^{-3} + d^{-4}$$

$$C_3(d) = [1 \;\; 1 \;\; 4 \;\; 4 \;\; 1] = d + d^{-1} + 4d^{-2} + 4d^{-3} + d^{-4}$$

The base five convolutional code was stored in the $n = N$ by L *code* array in the C structure *individual*. Fitness calculations were based on the candidate code's gmask syndrome.

- Associated Gmask - Once an individual's chromosome had been converted into the corresponding base five code parameters, base five table-based coding (discussed in Chapter 3) was used to determine the validity of the code. If the code was valid, the gmask of the code was formed. If the code was not invertible, an invalid flag was set. Invalid codes were not included in the initial population. For all generations following, if a code was invalid (i.e. non-invertible), all its parameters were set to a flag, $NaN = -99$, and its fitness was set to zero. To illustrate, using base five coding principles, the $n-k=2$ gmasks for the individual with binary chromosome

  $$111000001011100011000000111100$$

  are

  $$gmask_1 = [1\ \ 2\ \ 3\ \ 0\ \ 1\ \ 4\ \ 0\ \ 1\ \ 4]$$

  $$gmask_2 = [0\ \ 4\ \ 4\ \ 3\ \ 0\ \ 2\ \ 1\ \ 1\ \ 1]$$

  There were $n-k=2$ gmasks of length

  $$gLENGTH = w + n = 9$$

  where $w = 6$ was calculated as described in Chapter 3.

  Notice there are zeros in the gmask of the preceding example. A zero in a gmask implies that the base at that position is not involved in determining the syndrome. In a biological sense, this can be viewed as points where the 16S ribosomal RNA binds (for values greater than zero) or does not bind (for zero values) to the mRNA.

- Syndrome Distance Vector - After deriving the gmask, the next step towards fitness calculation was to find the syndrome vector by decoding the leader sequence (the received parity stream) in the training set. The received parity stream was composed of the thirty mRNA leader bases preceding the initiation (start) signal, the initiation signal (usually AUG), and twenty-seven bases from the coding region immediately following the initiation signal:

$$[b_{-30} \;\; b_{-29} \;\; ... \;\; b_{-1} \;\; A \;\; U \;\; G \;\; b_{+3} \;\; ... \;\; b_{+29}] \tag{4.5}$$

Bases numbered -30 to -1 belong to the leader (or 5' untranslated) region of the mRNA. Bases numbered +3 to +29 belong to the coding region of the mRNA. The A of the AUG initiation signal is position zero in the sequence. The ribosome covers approximately thirty bases of the mRNA at a time [9]. Therefore, a sixty base received parity sequence should be sufficient in representing which part of the mRNA is exposed to the gmask of the ribosomal decoder prior to the initiation of translation.

The syndrome, S, for a $gLENGTH$ subsequence occurring at position p (relative to position -30) of the received parity sequence was calculated as follows:

$$S(p) = \begin{bmatrix} P_p & P_{p+1} & ... & P_{p+gL-1} \end{bmatrix} \begin{bmatrix} gmask_{1,1} & ... & gmask_{2,1} \\ gmask_{1,2} & ... & gmask_{2,2} \\ ... & ... & ... \\ gmask_{1,gL} & ... & gmask_{2,gL} \end{bmatrix} \tag{4.6}$$

where $gL = gLENGTH$ and $S(p)$ was a one by $n-k$ vector of the form

$$S(p) = [S(p)_1, ..., S(p)_{n-k}]$$

To find the syndrome vector, for each shift of the gmask (the gmask shifts by n), we summed the $n-k$ syndromes produced at each shift position to get the syndrome vector $Svec$:

$$Svec(p) = \sum_{i=1}^{n-k} S(p)_i \tag{4.7}$$

where

$$p = 1, \quad ..., \quad nGSHIFTS$$

The constant $nGSHIFTS$ for horizontal code analysis (see section below discussing horizontal code analysis) was calculated as follows:

$$nGSHIFTS = \frac{RCV\_LENGTH - gLENGTH}{n} + 1 \tag{4.8}$$

where RCV_LENGTH was the length of the received parity mRNA sequence (60 in this work). For the vertical code models $nGSHIFTS = 1$.

Once $Svec$ was calculated for a given sequence and a given candidate code, the Hamming distance between the resulting syndrome vector and the all zero syndrome vector was calculated and stored in a syndrome distance vector, $Sdist$, of length $nGSHIFTS$. The C code below shows one method for calculating the syndrome vector and syndrome distance vector.

```
for (i=0; i<nGSHIFTS; i++)
    {
    Svec[i]=0;
    Sdist[i]=0;                 //default to 0
    for (j=0; j<ngmask; j++)    //ngmask=n-k
        {
        Svec[i]=Svec[i]+S[j][i];//S[j][i]=ith pos, jth gmask
        }                       //end j
    if (Svec[i]>0)              //non-zero syndrome
        {
        Sdist[i]=1;
        }
    }                           //end i
```

An example syndrome distance calculation:

- Given the gmask for the example individual and the following mRNA parity subsequence of length $gLENGTH$:

$$P_{mRNA} = [C \;\; G \;\; G \;\; C \;\; A \;\; A \;\; U \;\; A \;\; A \;\;]$$

- Convert P to its base five representation

$$P_{base5} = P = [3 \;\; 2 \;\; 2 \;\; 3 \;\; 1 \;\; 1 \;\; 4 \;\; 1 \;\; 1 \;\;]$$

- Using Equation 4.6, find the syndrome, S

$$S = [3 \;\; 3]$$

- From Equation 4.7, the syndrome vector, $Svec$, is

$$Svec = [1]$$

- The Hamming distance of this sequence from the all zero (one by one) vector is the syndrome vector, $Sdist$, which equals

$$Sdist = [1]$$

A non-zero syndrome in $Svec$ indicates an error in the received parity stream. An all-zero syndrome sequence indicates, in the general sense, that the candidate code associated with the current gmask produced the received mRNA parity sequence being considered. Deviations from a zero syndrome value indicate that the candidate code did not produce that particular subsequence of the mRNA parity sequence and this implies that an error occurred. Therefore, a Hamming distance comparison of the syndrome vector against the all-zero syndrome vector was a logical criterion for evaluating a code's fitness.

- Fitness Value - A fitness value was associated with each individual based on the performance of their gmask against the mRNA parity sequence being considered. The fitness for an individual sequence, f_{seq}, was calculated from the syndrome distance value:

$$f_{seq} = 1 - (\frac{1}{nGSHIFTS} \sum_{i=1}^{nGSHIFTS} Sdist_i) \tag{4.9}$$

  The higher the f_{seq} value, the better the individual solution.
- Scaled Fitness - The raw fitness value was scaled using $C_m = 1.5$ and Equations 4.2, 4.3, and 4.4. If a negative fitness occurred as a result of fitness scaling, the scaled fitness of the individual was set to zero. Scaling did not improve GA performance. Therefore, the unscaled fitness value was used to determine the target sampling rate (tsr) for each individual.
- Target Sampling Rate - Using the unscaled fitness value, the target sampling rate was set for each individual in the population using Equation 4.1.

Genetic Operators Used on Individuals

After each individual was analyzed, fitness statistics calculated, and target sampling rates determined, genetic operators were used to move the population towards the optimal solution. The three genetic operators used were selection, crossover and mutation.

Selection

Initially, fitness-proportional selection, or roulette-wheel selection, was used to select individuals for replication. The roulette-wheel method resulted in premature convergence to individuals in the local optima. Linear scaling was applied to the fitness values to prevent premature convergence and dominance of fit individuals early in the GA's run. Linear scaling did not solve the problem of premature convergence of the GA to the elite member of the population. Premature convergence prevented the GA from reaching high maximum fitness values. In an attempt to prevent premature convergence and encourage the GA to adequately explore the coding search space, target sampling rate selection was implemented.

The GA used the target sampling rate to determine the number of times an individual was allowed to mate. An individual's *tsr* was calculated using Equation 5.8:

$$tsr_i = 1 + \frac{f_i}{f_{avg}} \tag{4.10}$$

where f_i was the individual's fitness and f_{avg} was the average fitness for the generation.

If the GA used fitness scaling, an individual with zero fitness (including non-valid code candidates) could have a scaled fitness value greater than zero. This meant that individuals with zero unscaled fitness values theoretically still had a slight chance of mating. In theory this would preserve the diversity of the candidate population and since invalid codes could possibly recombine to form valid codes, this was positive for the GA. Target sampling rates were calculated using scaled and unscaled fitness values. Both methods performed well. The unscaled, fitness-based *tsr* achieved a fitness as high and. in an early run, higher than the scaled, fitness-based *tsr*. A few runs of the GA were used to compare the most effective method. Since initial findings did not conclusively show scaling to improve the GA performance, unscaled, fitness-based *tsr* values were used for selection.

Once the *tsr* was calculated, selection proceeded as previously described. Use of target sampling rate-based selection solved the premature convergence problem and encouraged the GA to explore and locate codes with high fitness values.

Crossover

This work used parameterized uniform crossover to produce new individuals for the next generation. Other crossover methods such as single-point and two-point crossover were tested and considered. Parameterized uniform crossover provided greater variation among the elite members of the population than other crossover methods. The crossover rate, P_C, was set to 0.65. So, crossover should occur sixty-five percent of the time. The crossover probability for a single locus, P_{locus}, on the binary chromosome was set to 0.5. P_{locus} set to fifty percent was not disruptive and did not prove negative to the GA as feared with non-parameterized uniform crossover. Given P_C and P_{locus}, the crossover mask was generated as follows:

```
for (k=0;k<lenBvec; k++)        //UNIFORM CROSSOVER
    {
    if (pcrossover<=CROSSOVER_RATE)
        coMask[0][k]=random(2); //select which parent bit comes from
    else
        coMask[0][k]=0;         //assign value to parent 1
    coMask[1][k]=add2[coMask[0][k]][1]; //make 2nd child
                                //complement of that in 1st child
                                //if child0 gets P0,child1 gets P1
    }                           //end k
```

In the above C-code, *add2* is a lookup table which implements binary addition. Using the crossover mask created above, *child*1 and *child*2, held in the variable *people*, which is an array of individuals, are produced and stored in the next generation array, *gnxt*:

```
for (j=0; j<lenBvec; j++)   //copy appropriate value into binary
                            //chromosome of children
    {
    gnxt->people[i].binvec[j]=g.people[matepair[coMask[0][j]]].binvec[j];
    gnxt->people[i+1].binvec[j]=g.people[matepair[coMask[1][j]]].binvec[j];
    }                       //end j
```

Mutation

After new individuals were produced using the crossover operator, each locus in an individual's binary chromosome was mutated at a rate of P_M. For the base five GA, the probability of a locus undergoing mutation was $P_M = 0.005$. Mutation was implemented as follows:

```
/*MUTATE takes the generation and mutates individual bits
    based on the specified mutation rate.
    VARS:   Input/Output - gnxt=pointer to the next generation
*/
int mutate(struct generation *gnxt) {
    int i,j,k;
    double pmutate;                     //prob for deciding if to mutate

    for (i=0; i<POP_SIZE; i++)          //loop through members of gnxt
        {
        for (j=0; j<lengthIndiv; j++)   //loop through each bit in indiv
          {
          pmutate=(double)rand()/(double)RAND_MAX;
          if (pmutate<=MUTATE_RATE)     //mutate if prob < mutation rate
            {                           //flip bit
            gnxt->people[i].binvec[j]=add2[gnxt->people[i].binvec[j]][1];
            }
          }                             //end j
        }                               //end i
    return(0);
};
```

This work used elitism to ensure the elite individual in the present generation was as fit or more fit than the elite individual of the previous generation. The GA was also tested without elitism. Although the average fitness of the generation increased without elitism, the elite individual was sometimes lost. As a result, the maximum fitness values decreased without elitism. Therefore, the use of elitism was preferred.

Fitness Evaluation

An individual's fitness, f_{seq}, described how well the candidate code modeled a particular sequence. The individual sequence fitness measure, f_{seq}, was calculated from the syndrome distance value and Equation 4.9.

To illustrate fitness calculation, assume the following *Sdist* vector result for a given sequence and a given individual(gSHIFTS=18):

$$Sdist = [0\ \ 1\ \ 0\ \ 1\ \ 1\ \ 0\ \ 1\ \ 1\ \ 0\ \ 0\ \ 0\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 0\ \ 0]$$

Using Equation 4.9, f_{seq} is

$$f_{seq} = 1 - (\frac{1}{18} * 10) = 0.44444$$

In the above example, there are eight zeros in the syndrome vector. This means eight subsequences out of eighteen were generated by this code (Later in this chapter we present a fitness measure that takes the location of the zero syndrome into account when calculating fitness). An all-zero syndrome would result in a fitness value of one. Conversely, a syndrome of non-zero values would have a fitness value of zero. The fitness value of individuals whose coding coefficients produce non-valid codes were set to zero.

A second fitness measure fit_{avg} extended the definition of f_{seq} to evaluate the performance of a candidate code over a set of mRNA parity sequences. Using Equation 4.9, fit_{avg} was defined as:

$$fit_{avg} = \frac{1}{nSEQ} \sum_{i=1}^{nSEQ} f_{seq_i} \tag{4.11}$$

In Equation 4.11, $nSEQ$ is the number of sequences being evaluated. When evaluating one sequence at a time $nSEQ$ was set to one and $fit_{avg} = f_{seq}$.

Motif-Based Fitness

The f_{seq} fitness measure assumed all positions in the messenger RNA were equally important for ribosomal binding hence for initiation. But research has shown that particular positions on the mRNA leader affect initiation more than other positions [56, 9, 8, 44]. Recent statistical analysis (presented in Chapter 5) compared the probability of a given $[M_1,\ M_2,\ M_3]$ binding motif occurring in valid *E. coli* leader regions to the probability of the binding motif occurring in non-translated intergenic sequences (see Figure 5.2). The statistical analysis clearly showed that alignment of the 16S rRNA at specific positions on the leader of mRNA was more favorable than alignment at other positions.

A hydrogen-binding, motif-based fitness metric was developed and implemented to find coding models that locate these key regions on the mRNA leader. The gmask of such a code could be functionally paralleled to the 16S rRNA. The calculation of fitness values using motifs was similar to that of f_{seq} in Equation 4.9:

$$f_{seq}^{motif} = 1 - \frac{1}{nGSHIFTS} \sum_{i=1}^{nGSHIFTS} Sdist_i * Mvec_i \tag{4.12}$$

where, $Mvec$ was the one by $nGSHIFTS$ motif vector of positional weight values between 0 and $nGSHIFTS$. For accurate fitness calculation, the following must hold:

$$\sum_{i=1}^{nGSHIFTS} Mvec_i = nGSHIFTS$$

The individual sequence fitness value f_{seq} was equal to the individual sequence motif-based fitness value, f_{seq}^{motif}, when the motif weight vector was defined as:

$$Mvec = [1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1\ \ ]$$

The GA searched for motif-based convolutional code models for translation initiation using the following motif weight vector:

$$\begin{aligned} Mvec \;=\; & [0.05\ \ 0.15\ \ 0.20\ \ 0.25\ \ 3.24\ \ 8.10\ \ 2.16\ \ 1.98\ \ 0.38 \\ & 0.38\ \ 0.38\ \ 0.38\ \ 0.125\ \ 0.10\ \ 0.05\ \ 0.05\ \ 0.0125\ \ 0.0125] \end{aligned} \tag{4.13}$$

The motif weight vector in Equation 4.13 was designed so that position -18 to -9 contained eight-six percent of the weight. The weighting distribution (based on the hydrogen-binding analysis results in Figure 5.2 and an ad-hoc ordering of binding ratio magnitudes) for regions in the sixty-base mRNA leader sequence is described in Table 4.3. Table 4.4 shows the relative position (zero represents alignment of gmask with first base of initiation codon) of each motif weight in Equation 4.13. Fitness values for motif-based codes were higher than for equal-weight codes. This is because, a zero at a key position can significantly inflate the fitness value of motif-based codes.

Table 4.3: Regional weighting distribution for motif-based fitness calculation

Gmask Alignment Position in mRNA	Approximate Weighting Percentage
-30 to -21	3.61
-18 to -9	86
-6 to +3	8.44
+6 to + 21	1.94

Table 4.4: Motif weights by position on mRNA leader

Position	**Weight**	**Percent**
-30	0.05	0.278
-27	0.15	0.833
-24	0.20	1.111
-21	0.25	1.389
-18	3.24	18.0
-15	8.10	45.0
-12	2.16	12.0
-9	1.98	11.0
-6	0.38	2.111
-3	0.38	2.111
0	0.38	2.111
+3	0.38	2.111
+6	0.125	0.694
+9	0.10	0.556
+12	0.05	0.278
+15	0.05	0.278
+18	0.0125	0.069
+21	0.0125	0.069

Directional Codes

There are several levels of coding as discussed in Chapter 1. Aside from various macromolecular levels of coding, at the nucleic acid level, there may exist directional codes: horizontal codes and vertical codes.

<u>Horizontal Codes</u>

The horizontal code model is based on the hypothesis that the regions or subsequences in a single mRNA leader are encoded using a single code. Therefore one gmask can be applied to the entire sequence. To search for horizontal codes, the $(n = 3, k = 1, m = 4)$ candidate code's nine-base gmask was applied across the entire sixty-base leader sequence. This resulted in $nGSHIFTS = 18$ possible zero positions. The GA searched for the best code that modeled each individual initiation region rather than the ensemble. The horizontal code GA used the individual sequence fitness measure, f_{seq}, defined in Equation 4.9 and the motif-based individual sequence fitness measure defined in Equation 4.12.

<u>Vertical Codes</u>

Vertical codes are positional codes. The assumption for the vertical coding model is that positional similarities exist among leader regions and these positional differences may be due to varied error-control codes used to protect each region within the mRNA leader. From a biological perspective, regions such as Shine-Dalgarno (translation), TATA boxes and Pribnow boxes (transcription) are known to exist [9, 56]. Also, consensus sequences have shown positional similarities in ribosome binding sites [44][45]. Given similar parity subsequences at particular locations in the leader of translated sequences, it is plausible to consider the existence of codes with gmasks that produce zero parity when placed at the same position in each leader sequence. This is what the vertical code model attempts to capture.

A GA was designed to search for viable vertical codes. The $(n = 3, k = 1, m = 4)$ candidate code's nine-base gmask was applied against all $nSEQ$ mRNA leader sequences in the training set. The average sequence fitness measure, fit_{avg}, defined in Equation 4.11, was used to access the fitness of the candidate vertical code.

4.3.2 Procedure

The base-five coding operations were written as a library of error-control coding operations, *codeOps.c*. The genetic algorithm program, *gaOps.c*, used the base-five code operations library to implement the horizontal and vertical code searches.

Messenger RNA leader sequences from *E. coli* K-12 genome were used as training sequences for finding the best candidate code model. The *E. coli* mRNA sequence set used in this work was the same set parsed (from the original GenBank files) for verifiable translation and used by Rosnick [65]. There were 531 sequences in Rosnick's sequence set. Every even sequence (where the first sequence in the file is considered sequence zero) was designated as part of the training set. There were 266 *E. coli* leader sequences in the training set used to evaluate candidate codes.

The GA executed $RUNS\_PER\_SEQ$ times. The number of runs per sequence or per position depended on the code model (horizontal or vertical) the GA was trying to locate. Multiple GA runs ensured exploration from various starting points in the search space. Since the initial population was randomly generated, multiple runs should produce different sets of "starting point" populations and locate several possible optima codes. The number of runs per sequence/position was based on the total run time of the GA and the size of the solution space. The search algorithm executed on a 600 MHz, Microsoft Windows 2000 system with the following parameters for the horizontal and vertical code GAs:

- Horizontal Table-Based Codes - The GA searched for horizontal codes that best described each of the 266 leader sequences in the training set. For the horizontal code GA,

  $$POPULATION\_SIZE = 750$$

  $$NUMBER\_GENERARIONS = 1010$$

  $$RUNS\_PER\_SEQ = 15$$

  The total time to execute the GA for all 266 sequences was approximately 79 hours. The motif-based horizontal GA used the same parameters and executed in about the same amount of time.

- Vertical Table-Based Codes - The GA searched for vertical codes for each of the possible $nGSHIFTs = 18$ positions on the sixty-base leader sequence. The fitness was based on the average fitness over all of the 266 *E. coli* leader sequences in the training set. For the vertical code GA,

  $$POPULATION\_SIZE = 1200$$

  $$NUMBER\_GENERARIONS = 2000$$

  $$RUNS\_PER\_SEQ = 20$$

  The parameters for the vertical GA were larger than the horizontal GA because the vertical GA was searching for eighteen code models while the horizontal GA was searching for 266 code models. The total time to execute the vertical GA for all 18 positions was approximately 72 hours.

The GA produced three output files:

Table 4.5: Contents of summary data file

Column	**Data**
1	Sequence identity (all even numerals) or Position Number (Vertical Codes)
2	Run Number - which run (out of $RUNS\_PER\_SEQ$) the best code occurred
3	Generation Number - which generation best code occurred; always equal to $NUMBER\_GENERATIONS$ since GA uses elitism
4	Maximum Fitness - the fitness of the best code; also the maximum fitness for the generation
5	Minimum Fitness - the lowest fitness for the generation
6	Average Fitness - the average fitness for the generation
7 to 11	Coefficients for generator 1, C_1, of the best code
12 to 16	Coefficients for generator 2, C_2, of the best code
17 to 21	Coefficients for generator 3, C_3, of the best code
22 to 30	Coefficients for the nine-base gmask1 for the best code
31 to 39	Coefficients for the nine-base gmask2 for the best code
40 to 57(horiz) / 40(vert)	Syndrome distance pattern for the best code

- Summary File - A Matlab formatted space delimited data file. Each row of the summary file contained information regarding the elite code candidate (based on individual sequence fitness) for each sequence. The elite code was selected from all populations produced over $RUNS\_PER\_SEQ$ runs. So, for horizontal codes the summary data file contained 266 rows and for vertical codes the summary data file contained 18 rows. For a given row, Table 4.5 defines the information stored in each column.
- All File - A LOG (text) comma-delimited file containing the same information as the summary file (see Table 4.5) for the best code per run, per sequence. The all log file had $RUNS\_PER\_SEQ * 266$ rows for the horizontal GA and $RUNS\_PER\_SEQ * 18$ rows for the vertical GA. In the all file the coefficients

Table 4.6: Contents of the all log file

Column	**Data**
1	Sequence identity (all even numerals) or Position Number (Vertical Codes)
2	Run Number
3	Generation Number - which generation best code occurred; always equal to $NUMBER\_GENERATIONS$ since GA uses elitism
4	Maximum Fitness - the fitness of the best code; also the maximum fitness for the generation
5	Minimum Fitness - the lowest fitness for the generation
6	Average Fitness - the average fitness for the generation
7	C_1 of the best code
8	C_2 of the best code
9	C_3 of the best code
10	Gmask for the best code
11	Syndrome distance pattern for the best code

of the generators, gmasks, and syndrome distance vector were written as one string. Instead of five comma separated numerals representing generator one, there will be a single, five-numeral string (a concatenation of the five generator one coding coefficients) representing C_1. Table 4.6 defines the columns of the all file.

- Population File - A LOG file that stored the details of the GA run including: leader sequences in the input file, begin and end times for GA run, GA parameters, fitness motif weight vectors, and detailed information about individual codes with fitness values above the fitness threshold score (45% for non-motif horizontal codes, 90% for motif-based horizontal codes, and 45% for vertical codes).

The output of the GAs were analyzed; results are presented in the following sections.

4.3.3 Results

The previously described genetic algorithm was used to search for base five, table-based convolutional codes which produced the mRNA leader sequences in the *E. coli* training or model set. Results from the summary data files for the horizontal, horizontal motif, and vertical convolutional code models were analyzed using analysis routines developed with *Matlab*. Results of the analysis follow.

The horizontal, equal-weight genetic algorithm searched for the optimal, convolutional code for a single sequence. All of the syndrome positions carried the same weight for fitness calculations. The motif-based GA search produced codes with unequal weight distribution, hence unequal error protection for various positions in the leader sequence. The GA search for optimal vertical searched for the optimal convolutional code for each position of the leader region. The syndrome distance vector for each code indicates whether the associated decoder recognizes the subsequence at hand. If the genetic algorithm found the perfect code - the convolutional coding system that produced the exact sequence - then the syndrome distance vector would be the all zero vector and the fitness value would be one. Figure 4.1 shows the average syndrome distance value for the optimal codes discovered using the *E. coli* model set. In Figure 4.1 the horizontal axis is position relative to the first base in the initiation codon and the vertical axis is the average syndrome distance value. For the horizontal codes the individual syndrome distance values for each code model are averaged over the 266 models. The vertical code model is the average syndrome value for each of the 18 positional models discovered in the GA search. The graph of the average syndrome distance for the vertical code models does not indicate any regions of significant activity. But, closer analysis of the vertical code results, depicted in Figure 4.2, indicate the vertical code model does behave relatively different over various regions. The lowest average syn-

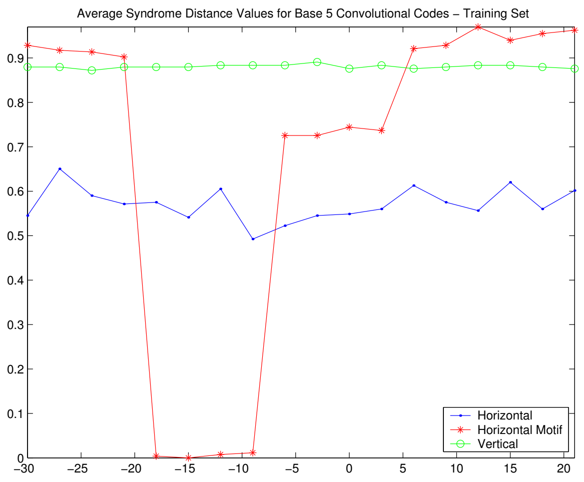


Figure 4.1: Average Syndrome Distance for Base Five Table-Based Convolutional Code Models for Translation Initiation

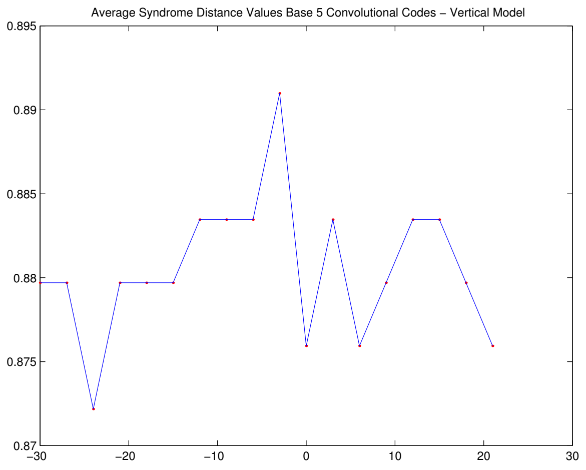


Figure 4.2: Average Syndrome Distance From the All-Zero Syndrome for Base Five Table-Based Convolutional Code Models - Vertical Models

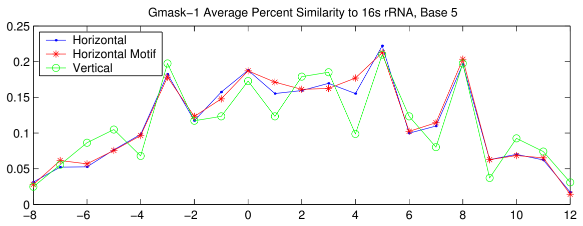
drome distance value for the equal-weight horizontal code model occurs at position -9 while the motif-based horizontal code models have approximately zero average syndrome distance values from position -18 to position -9, key positions in the translation initiation process.

The principle hypothesis of this work models the mRNA as a noisy encoded signal and the ribosome to a convolutional decoder. Given this parallelism, the gmask of the code models for translation initiation may bear some similarity to the exposed part of the 16S rRNA. To test this, the nine base gmask of each code was compared to the last thirteen bases of the 16S rRNA at all possible alignment positions from -8 to +12. If the coefficients of the gmask are numbered (left to right) from one to nine and the last thirteen bases of the 16S rRNA are numbered (3' to 5') from one to thirteen, then alignment position -8 represents alignment of base one of the exposed 16S rRNA subsequence with coefficient nine of the gmask. Alignment position +12 represents the last base of the 16S rRNA thirteen base subsequence aligned with the first coefficient of the gmask. The gmask is shifted to the right by one and similarity is measured for each alignment position. Similarity values are calculated over the overlapping gmask and 16S rRNA subsequence using Equation 4.14:

$$Similarity = 1 - d_H(gmask, 16s\_rRNA) \tag{4.14}$$

In Equation 4.14, d_H indicates the Hamming distance measure.

Figure 4.3 shows the average similarity between the gmasks of the model codes and the last thirteen bases of the 16s rRNA sequence. The horizontal axis indicates position as it relates to correlational analysis, where zero represents the initial full overlap point between the 16S rRNA and the gmasks. The vertical axis is the average percent similarity between the nine-base gmask and subsequent nine-base subsequences of the exposed region of the 16s rRNA. Although the vertical code's average syndrome distance values were higher (i.e. lower fitness) than the horizontal models, its average percent similarity is comparable to



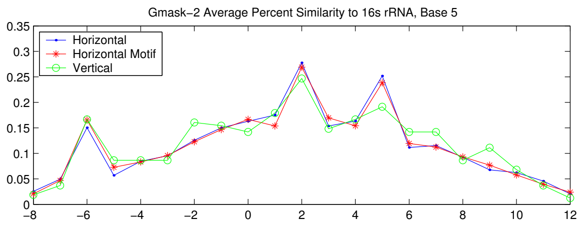


Figure 4.3: Average Similarity Between Exposed Part of the 16S rRNA and the Gmasks of Base Five Table-Based Convolutional Code Models for Translation Initiation

the horizontal code models. For gmask1, the highest average percent similarity occurs at alignment position +5 where the overlapping 16s subsequence is

$$\ldots\ U\ \ C\ \ C\ \ A\ \ C\ \ U\ \ A\ \ G\ ..$$

The models' gmask2 reached a slightly higher percent similarity value. This occurs at alignment position +2 where the overlapping 16s rRNA subsequence is

$$\ldots\ U\ \ C\ \ C\ \ U\ \ C\ \ C\ \ A\ \ C\ \ U\ ..$$

Convolutional codes with high similarity to the last thirteen bases of the 16S rRNA and low syndrome distance (i.e. high fitness) values would, from a biological perspective, be more plausible models for translation initiation. The relationship between a code's gmask percent similarity to the exposed part of the 16S rRNA and the code's fitness value was analyzed. Figure 4.4 depicts the relationship between each code models fitness score and similarity value. The horizontal axis in Figure 4.4 is fitness and the vertical axis is percent similarity. For gmask1, the equal-weight horizontal code model and the vertical code model

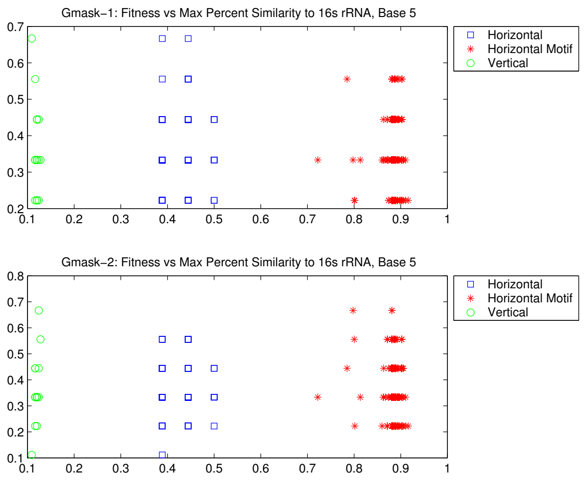


Figure 4.4: Individual Fitness versus Individual Similarity Values for Base Five Code Models

achieve the highest percent similarity. But, for gmask2 the motif-based horizontal code model and the vertical code model achieve the highest percent similarity scores. In all code model groups, there exists individuals with high similarity values and relatively high fitness.

4.3.4 Discussion

The vertical code GA searched for the best code, per position, for all 266 leader sequences in the testing set. Therefore, the syndrome distance values of the vertical code model is expected to be larger than the distance values for the horizontal code models which are optimized for a single leader sequence. This work expected the vertical code model's syndrome distance to noticeably decrease within the Shine-Dalgarno and initiation region (between position -15 and 0). The lowest average distance for the vertical code model occurred around -24, 0, and +6. So the vertical model detects the initiation codon but not the non-random or Shine-Dalgarno domain.

Although there was a slight drop in average syndrome distance at -15 and -9, this work expected that the equal-weight horizontal code model would contain more positions close to zero syndrome since the horizontal code GAs optimized for individual sequences. The equal-weight horizontal code does achieve a minimum syndrome distance value at 0, indicating recognition of the initiation region. The results of the motif-based horizontal code model did produce zero and almost zero syndrome distance values in the -18 to -9 region. The motif-based code model indicates recognition of the non-random domain and the Shine-Dalgarno region. Vertical codes can be approximated using motif-based horizontal codes and assigning extremely high weighting coefficients to positions of interest. The motif-based horizontal code results lend credibility to the coding theory view of translation initiation.

The existence of individual gmask values with high fitness (relative to the coding model group) and high similarity to the exposed portion of the 16S rRNA are encouraging. The similarity analysis may have been affected by the gmask formation technique for base five table based coding. Since actual syndrome values are mapped to a $n-k$ bit representation and other mask values are generated based on an initial basis mapping, the similarity analysis may vary when using other mask-based convolutional decoding approaches.

The behavior of the base five models are encouraging and require further investigation. In the next chapter, code models that capture the functional aspect of the translation initiation are explored.

Chapter 5

Constructing Codes from Binding Motifs

Studies of prokaryotic translation initiation sites reveal that ribosomal binding sites appear to evolve to functional requirements rather than to genetic sequences that produce the strongest binding site [10]. Several factors influence translation of mRNA sequences, including: initiation codon, presence and location of the Shine-Dalgarno sequence, spacing between the initiation codon and the Shine-Dalgarno domain, the second codon following the initiator codon, and possibly other nucleotides in the -20 to +13 region of the mRNA leader region [8]. These factors influence how the small subunit of the ribosome interacts with the mRNA leader region such that conditions are favorable for successful translation initiation. A key factor in translation and other genetic processes is binding of macromolecules to and interaction with specific sites on nucleic acid sequences or other molecules. Based on this, translation initiation can be modeled by analyzing possible binding patterns between mRNA and the exposed portion of the 16S rRNA.

In the sections that follow, a binding analysis of translation initiation sites is presented. Binding analysis data are used as inputs to a genetic algorithm which searches for the best binary binding codes to model the *E. coli* translation initiation system. The chapter concludes with an evaluation of the candidate codes discovered.

5.1 Binding Analysis of Translation Initiation Sites

The exposed part of the 16S rRNA (the last 13 bases of the 16s) is available for base pairing (binding) with the mRNA [9, 56, 8]. On average, five nucleotides on the mRNA leader complement pair with the exposed part of the 16S rRNA [8]. The average distance between the 16S rRNA binding region and the initiation codon is seven nucleotides [8]. The binding pattern formed between the 16S rRNA and the mRNA leader region directly affects translation initiation. Although binding is related to higher level interactions influenced by mRNA structure, rRNA structure, and ribosomal and protein interactions, this work is based on the hypothesis that translation initiation can be viewed from a binary perspective.

5.1.1 Functional Definition of mRNA Leader and Ribosomal Interaction

Although a leader sequence with perfect complementary base pairing to the 16S rRNA may not be the most viable sequence from an evolutionary view point, it is plausible to assume that increased affinity to the 16S rRNA increases initiation potential. Not only must a leader sequence contain nucleotides that bind to the 16s, the binding must occur within a reasonable proximity to the initiation codon. These requirements have been discussed in [8]. Since translation initiation is influenced by positional binding, the biological process of translation initiation can be mapped to a functional domain. Using sequence information and the last 13 bases of the 16S rRNA,

$$3' \; A \; U \; U \quad C \; C \; U \quad C \; C \; A \quad C \; U \; A \quad G \; ...5'$$

mRNA leader regions can be mapped to positional binding representations.

For each mRNA leader sequence, the functional mapping process is as follows:

1. For each position, p, in the mRNA sequence, align the 3' end of the last thirteen bases of the 16S with base p to base $p + 12$.

2. Assign a 1 to each position where the aligned leader and rRNA sequence complement pair (when using base five RNA representation, this is where the base five summation is zero) and a 0 to overlapping base positions that do not complement pair.

3. Convert the thirteen base binary vector to a decimal number between 0 and $2^{13}-1$, where the leftmost digit is the most significant bit.

4. The decimal integer in position p represents the thirteen bit functional relationship between the exposed portion of the 16S rRNA and the thirteen-base mRNA subsequence at position p.

5. Repeat this process for the remaining positions. There will be

$$NumValid = Length \;\; mRNA - Length \;\; Exposed \;\; Part \;\; 16s \;\; rRNA + 1$$

valid positions per sequence. Each mRNA sequence evaluated was sixty bases long. Therefore $NumValid = 60 - 13 + 1 = 48$.

To illustrate the mapping process, consider the following 15 base mRNA leader subsequence:

$$UAU \;\; AGG \;\; AGG \;\; CGG \;\; AUG$$

. For the above sequence, $NumValid = 15 - 13 + 1 = 3$. The three binary binding vectors for position 1 to $NumValid = 3$ are:

$$BinaryBindingVec_1 = 1\;\;1\;\;0\;\;0\;\;1\;\;0\;\;0\;\;1\;\;0\;\;0\;\;0\;\;0\;\;0$$

$$BinaryBindingVec_2 = 0\;\;0\;\;1\;\;1\;\;1\;\;1\;\;1\;\;1\;\;0\;\;1\;\;0\;\;0\;\;0$$

$$BinaryBindingVec_3 = 1\;\;1\;\;0\;\;1\;\;0\;\;0\;\;1\;\;0\;\;0\;\;1\;\;1\;\;1\;\;0$$

5.1.2 Binding Pattern Representation

After mapping the mRNA sequence into binary binding vectors, each vector is classified based on their $(M1, M2, M3)$ binding pattern values. Given a thirteen base binary binding pattern, the value $M1$ is the greatest number of consecutive base pairings (1's), $M2$ is the second greatest and $M3$ is the third greatest. The expectation is that binding patterns with large $M1$ values, within an acceptable distance from the initiation codon, will favor translation initiation. Sequences with smaller $M1$ values would be expected to have significant $M2$ and $M3$ values to increase the probability of ribosome binding.

Each positional binary binding pattern was classified based on their $(M1, M2, M3)$ value. Different binary binding patterns can belong to the same $(M1, M2, M3)$ class. Each $(M1, M2, M3)$ class was assigned a number between 1, (M1=13, M2=0, M3=0) and 91, (M1=0, M2=0, M3=0). For example, given the following two binary binding vectors:

$$BinaryBindingVec_A = 1\ \ 1\ \ 1\ \ 0\ \ 0\ \ 0\ \ 1\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0$$

$$BinaryBindingVec_B = 0\ \ 1\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 1\ \ 1\ \ 1\ \ 0$$

Both vectors would be classified as (M1=3, M2=1, M3=0), or with the classification number 80.

The probability of each classification number occurring (based on all possible thirteen base binding vectors) was calculated and classification thresholds were tabulated.

5.1.3 Binding Analysis Applied to Leader and Non-leader mRNA Sequences

A hypothesis of this work is that valid initiation regions fall within a $(M1, M2, M3)$ pattern threshold while most non-leader sequences do not. To investigate the plausibility of the hypothesis, binding analysis was performed on a set of 531 *E. coli* leader sequences, 1000

E.coli intergenic, non-leader sequence, and 1000 randomly generated sequences. The non-leader and random sequences all had AUG initiation sites in the center of the "candidate" sequence. Each sequence contained 60 nucleotide bases represented in base five.

Each sequence in the three sequence sets was mapped to their functional equivalent binary binding vector and classified based on their $(M1, M2, M3)$ values as previously described. The positional $(M1, M2, M3)$ vector was evaluated for each sequence group per position. Figure 5.1 shows the percent of sequences with (M1=4, M2=0, M3=0) or stronger binding pattern per position. In Figure 5.1, the horizontal axis represents position and the

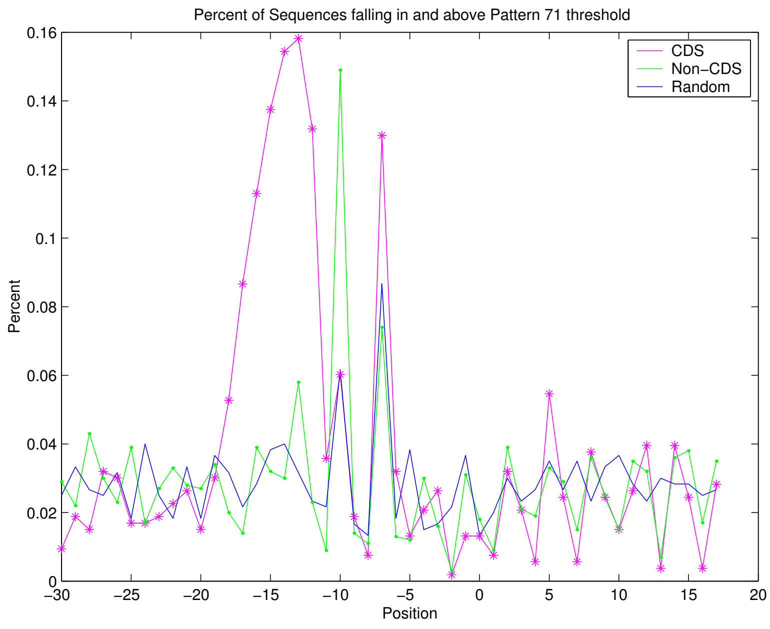


Figure 5.1: Percent of Sequences with Binding Pattern of 4,0,0 and Above

vertical axis represents the percent of sequences in each sequence group with a binding pattern of (M1=4, M2=0, M3=0) or stronger. From Figure 5.1 we note the following:

- The region between -18 and -9 has the greatest distinction between the translated sequence group and the other non-translated sequence groups. This is consistent with regions of distinction found in previous work [11]. The percentages for the

Table 5.1: Distribution of Strongest Binding Patterns for Translated, Non-translated, and Random Sequence Groups

$(M1, M2, M3)$	Translated (%)	Non-translated (%)	Random (%)
13,0,0 to 6,0,0	14.12	1.50	1.50
5,x,x	22.41	3.80	3.67
4,x,x	30.51	14.00	14.50
3,x,x	27.68	39.20	40.67
2,x,x to 0,0,0	5.27	41.50	39.67

translated sequences in the -18 to -9 region is greater than (up to five times greater) the threshold value of 3.013%

- Inside the coding region for translated sequences (position 0 and greater), there is a clear synchronization pattern which repeats every three bases. This pattern is not as consistent in the non-translated nor the random sequence groups.

The statistics depicted in Figure 5.1 do not reflect the best binding pattern but looks at all occurring binding patterns. Using the same sequence groups, the best binding pattern (lowest classification number) was selected for each sequence in each group. Given the results from Figure 5.1, the strongest binding was analyzed for positions -18 to -12. Table 5.1 shows the distribution of strongest binding patterns in each of the $(M1, M2, M3)$ binding classification groups. From Table 5.1 a binding pattern threshold of 71 or (M1=4, M2=0, M3=0) or stronger captures a large amount of translated sequences while excluding a significant number of non-translated and random sequences.

The functional binding statistics have thus far characterized the binding behavior over specific position ranges. The key to defining the binary binding model (and ultimately the convolutional coding model) for translation initiation lies in the ability to capture positional binding information. Ribosomal recognition of a protein translation initiation site does not depend on only one position in the leader sequence. Understanding positional binding will lead to a more knowledgeable model for ribosome-mRNA interaction for initiation. Individual sequence group positional binding information is not sufficient for binding vector model construction. Therefore, similar to statistics usable for sequence classification purposes, this work compared positional binding information for translated sequences versus non-translated sequences using the following joint probability ratio.

$$p = \frac{P(Bind, Position|Translated)}{P(Bind, Position|Non-translated} \tag{5.1}$$

The probability was calculated for positions -18 to -3 and the results (by binding pattern classification groups) are shown in Figure 5.2 In Figure 5.2, the horizontal axis is position

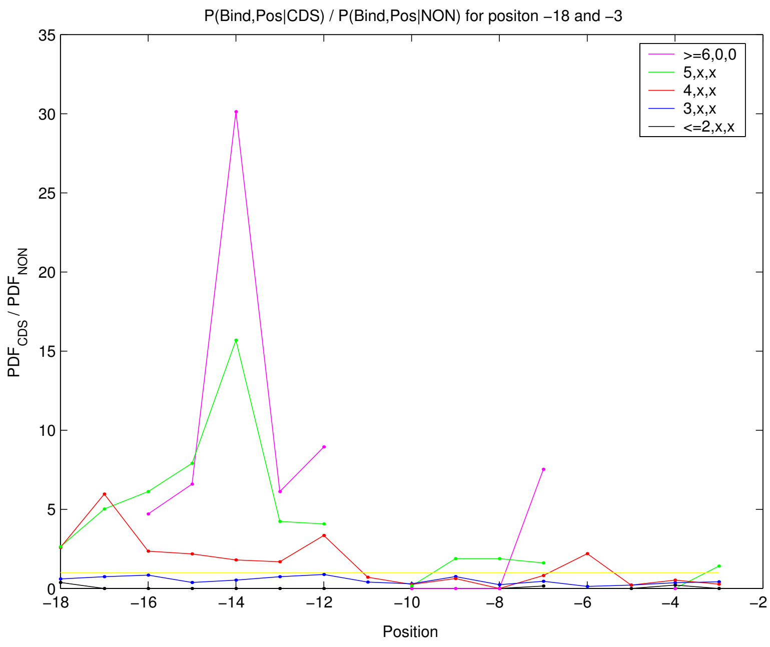


Figure 5.2: Positional Binding Ratio of Translated Sequence Group to Non-Translated Sequence Group

relative to the first base of the initiation codon and the vertical axis is the ratio defined in Equation 5.1. Discontinuities in Figure 5.2 are a result of divide by zeros. Ratios above

Table 5.2: Location of Largest Translated to Non-translated Positional Binding Ratio Value

$(M1, M2, M3)$	Position
13,0,0 to 6,0,0	-14
5,x,x	-14
4,x,x	-17
3,x,x	Ratio less than 1
2,x,x to 0,0,0	Ratio less than 1

one indicate positions where translated sequence binding dominates non-translated. Ratios less than one indicate the opposite occurrence. Table 5.2 summarizes the key positions for each binding classification group in Figure 5.2 that achieved ratios greater than one. Positional ratio values are used to develop horizontal motif-based convolutional codes for binary binding vectors (also used to develop horizontal motif-based base five convolution codes in the previous chapter).

5.1.4 From Binding Vectors to Codewords

Each binding vector pattern can be considered a codeword for that position. The question becomes what coding system produced the binding vector codewords and is the coding system a horizontal encoder/decoder or a vertical coding scheme? The discovery of effective binary convolutional code models that depict the functional behavior of the ribosome-mRNA interaction will demonstrate that information theory, coding theory specifically, principles can be applied to functional models of genetic regulatory systems. From binary code models of initiation, we can define functional heuristics, correct errors that lead to system malfunction, and possibly define more efficient functional heuristics which translate into changes in a gene's leader sequence.

5.2 Genetic Algorithms for Binding-Based Translation Initiation Codes

The thirteen-bit binding patterns present in translated sequences are viewed as codewords generated by a candidate convolutional encoder. Discovering the "best" binding codes can serve as a foundation for defining valid leader sequences and key positions on the mRNA leader region that positively or negatively affect translation initiation.

As previously mentioned, most convolutional code construction methods rely on computer search techniques such as genetic algorithms [63, 60]. Similar to the GA used in Chapter 4 for constructing optimal table-based base five convolutional codes for translation initiation, the section which follows describes the use of genetic algorithms for constructing binary table-based convolutional codes for translational initiation. An optimal code should be able to recognize the binary binding patterns which describe the functional relationship between the ribosome and the mRNA.

5.2.1 Methodology

As described in Chapter 4, the effectiveness of the functional convolutional code was evaluated using the $n - k$ gmasks constructed from the candidate code (see Chapter 3 Section 3.2for a description of binary table-based coding and gmask construction). The GAs search space included all possible codes for a given (n, k, L) binary convolutional code. The GAs population, potential solution space, fitness evaluation method, and genetic operators were defined based on the objective: locate a $(n = 3, k = 1, m = 4), L = 5$ binary convolutional code that has the greatest probability of producing the binary binding vector for each *E. coli* leader sequence in the training set. Elements of the GA (as defined in Chapter 4) for binary binding codes are described.

Individuals

Similar to the GA presented in the previous chapter, the GA used to find "good" binary binding codes was a multi-parameter genetic algorithm. For the (3,1,4) candidate codes, there were $n = 3$ generators each with $L = 5$ binary coefficients. Each candidate code (or individual) was represented as a concatenation of $n * L = 15$ coefficients. There were two possible values for each coefficient in the three generators, 0 or 1. Therefore each coefficient or parameter can be represented using $B_{param_i} = B_{param} = 1$ bit for all i. For the (3,1,4) candidate codes, there were a total of $n * L * B_{param} = 15$ bits in the binary chromosome representation of each code or individual. The search space contained a total of $2^{n*L*B_{param}} = 2^{15} = 32768$ candidate codes. Since the search space was relatively small, an exhaustive search could have been used to locate the best binary code for each sequence. Following the methods from the previous chapter, this work elected to use a genetic algorithm to search for the best binary code since as n and L increase, a GA search could prove more efficient.

An individual was defined with the same eight elements as listed in the *struct individual* definition for individuals in the base five GA (described in Chapter 4). Each individual structure was composed of the following:

- Binary Chromosome - the fifteen bit representation of the code. The genetic operators used the binary chromosome to produce new and diverse individuals for the next generation. The initial population was constructed by randomly generating $N = POPULATION\_SIZE$ binary chromosomes using the sample C code, *mkindividual*, listed in Chapter 4. For the (3,1,4) code, an example $lengthIndiv = n * L * B_{param} = 15$ bit binary chromosome is:

  011111011000011

- Decimal Tag - The base ten integer representation of the candidate code in binary chromosome. The value for the decimal tag ranged from 0 to $2^{nL} - 1$.
- Candidate Convolutional Code - Binary chromosomes were converted into coding coefficients for candidate codes. Unlike the GA for base five codes, chromosome to code conversion did not require intermediate steps. The first L bits of the binary chromosome were assigned to the first binary code generator. The next L bits of the binary chromosome were assigned to the second generator, and so forth till all n generator vectors were defined.

  For the example binary chromosome above, taking every $L = 5$ bits to represent the generator coefficients for C_1 through C_n, where $n = 3$, we get the following genetic convolutional code:

$$C_1(d) = [0 \ \ 1 \ \ 1 \ \ 1 \ \ 1] = d^{-1} + d^{-2} + d^{-3} + d^{-4}$$

$$C_2(d) = [1 \ \ 0 \ \ 1 \ \ 1 \ \ 0] = d + d^{-2} + d^{-3}$$

$$C_3(d) = [0 \ \ 0 \ \ 0 \ \ 1 \ \ 1] = d^{-3} + d^{-4}$$

  The binary convolutional code was stored in the $N = n$ by L *code* array in the C structure *individual*. Fitness calculations were based on the candidate code's gmask syndrome.
- Associated Gmask - Using binary table-based coding (as discussed in Chapter 3), the code's gmask was used to determine whether the candidate convolutional code was valid. The associated gmask was found for all valid codes. Non-invertible codes resulted in an invalid flag being set and were excluded from the initial population. As in the base five case, for all generations following, if a code was invalid (i.e. non-invertible), all its parameters were set to a flag,

$NaN = -99$, and its fitness was set to zero. Using binary, table-based coding techniques, the associated $n-k=2$ gmasks for the individual with binary chromosome

$$011111011000011$$

are as follows (each gmask is of length $gLENGTH = w + n = 9$):

$$gmask_1 = [1\ \ 0\ \ 0\ \ 0\ \ 1\ \ 1\ \ 0\ \ 1\ \ 0]$$

$$gmask_2 = [0\ \ 0\ \ 1\ \ 0\ \ 0\ \ 0\ \ 1\ \ 0\ \ 1]$$

As in base five codes, zeros in the gmask imply that the binding at that position was not involved in determining the syndrome; hence the base in that position may not be involved in determining the syndrome value for that particular ribosome/mRNA alignment.

- Syndrome Distance Vector - Using the associated gmasks, the syndrome vector for the binding pattern of each leader sequence in the training set was calculated. The leader sequence input was the $NumValid = 60 - 13 + 1 = 48$ decimal values that represent the thirteen base binary binding vector (discussed in earlier sections) at each valid alignment position, p, where $p = -30... + 17$. The leader sequence input was of the form:

$$[bp_{-30}\ \ bp_{-29}\ \ ...\ \ bp_0\ \ bp_{+1}\ \ bp_{+2}\ \ ...bp_{+17}]$$

where bp_p was the decimal representation for the thirteen bit binding pattern that formed when the exposed part of the 16S rRNA aligned with the mRNA at position p. Each decimal value was converted to a corresponding thirteen bit binary binding vector; this thirteen bit vector was the received parity sequence for position p of the leader sequence. The following is an example leader sequence decimal binding vector:

$$[1046\ \ 6144\ \ 0\ \ 24\ \ 12\ \ 164\ \ 4194\ \ 74\ \ ...\ \ 80\ \ 272\ \ 772\ \ ]$$

The binary binding vector for $bp_{30} = 1046$, the received thirteen-bit sequence, is

$$r_{30} = [0\ \ 0\ \ 1\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 1\ \ 0\ \ 1\ \ 1\ \ 0]$$

Given a thirteen bit received sequence and a gmask length of $gLENGTH = w + n = 9$, the number of possible times the gmask can shift over the thirteen base sequence (producing $n - k = 2$ syndrome bits per shift) was defined as

$$nGSHIFTS = \frac{RCV\_LENGTH - gLENGTH}{n} + 1 \tag{5.2}$$

where RCV_LENGTH was the length of the received parity binding sequence. The value of $nGSHIFTS$ was two, hence the gmask was applied twice. Starting with the first bit in the binding vector, the gmask was applied, shifted by $n = 3$ and applied again beginning with the fourth bit of the binary binding vector. Although the actual length of the binary binding vector was thirteen, this work only used the first twelve bits to calculate the syndrome. Therefore the actual RCV_LENGTH value was twelve.

For each decimal binding value, bp_p in the mRNA leader sequence, there were $nGSHIFTS = 2$, $gLENGTH$ binary subsequences used to determine the syndrome, S, for position p. The syndrome sequence $S(p)^i$ for each leader sequence, where $i = 1 \;..\; nGSHIFTS$, was calculated as follows:

$$S(p) = \begin{bmatrix} gmask_{1,1} & : & gmask_{1,2} \\ gmask_{2,1} & : & gmask_{2,2} \\ : & : & : \\ gmask_{n-k,1} & : & gmask_{n-k,gL} \end{bmatrix} \begin{bmatrix} r(1)_p^1 & : & r(1)_p^{nGHSHIFTS} \\ r(2)_p^1 & : & r(2)_p^{nGSHIFTS} \\ : & : & : \\ r(gL)_p^1 & ... & r(gL)_p^{nGSHIFTS} \end{bmatrix} \tag{5.3}$$

where each column in the received vector, r, is the $gL = gLENGTH$ received binary binding subsequence for position p and S is a $n-k$ by $NumValid *$ $nGHSIFTS$ matrix of the form

$$S = \begin{bmatrix} S(-30)_1^1 & : & S(-30)_{n-k}^{nGSHIFTS} & : & S(+17)_1^1 & : & S(+17)_{n-k}^{nGSHIFTS} \\ S(-30)_2^1 & : & S(-30)_{n-k}^{nGSHIFTS} & : & S(+17)_2^1 & : & S(+17)_{n-k}^{nGSHIFTS} \\ : & : & : & : & : & : & : \\ S(-30)_{n-k}^1 & : & S(-30)_{n-k}^{nGSHIFTS} & : & S(+17)_{n-k}^1 & : & S(+17)_{n-k}^{nGSHIFTS} \end{bmatrix} \tag{5.4}$$

Given the syndrome matrix, S, the syndrome vector $Svec$ was formed by summing the columns of the syndrome matrix (summing all syndrome values for each position):

$$Svec(p)^i = \sum_{j=1}^{n-k} S(p)_j^i \tag{5.5}$$

where

$$p = [1, \;\; ..., \;\; NumValid] \qquad and \qquad i = [1, \;\; ..., \;\; nGSHIFTS]$$

The syndrome vector was a 1 by $NumValid * nGHSIFTS = 96$ vector used to determine the Hamming distance, $Sdist$, of the syndrome from the all-zero syndrome vector of equal dimension.

As stated in earlier chapters, an all-zero syndrome indicates that the candidate code produced the binding vector pattern. Deviations from the all-zero $Svec$ decreases the probability that the candidate code associated with the gmasks produced the binary binding vector being evaluated. Therefore the Hamming distance measure against the all zero syndrome was a logical fitness criterion.

- Fitness Value - Similar to the base five GA, the fitness for an individual, f_{seq}, was calculated from the syndrome distance value:

$$f_{seq} = 1 - (\frac{1}{NumValid * nGSHIFTS} \sum_{i=1}^{NumValid*nGSHIFTS} Sdist_i) \qquad (5.6)$$

  The higher the f_{seq} value the better the candidate code.

  The fitness measure fit_{avg} extended the definition of f_{seq} to evaluate the performance of a candidate code over a set of mRNA parity sequences. This fitness measure is described in Equation 5.7:

$$fit_{avg} = \frac{1}{nSEQ} \sum_{i=1}^{nSEQ} f_{seq \quad i} \qquad (5.7)$$

  The candidate codes for the horizontal coding model used the f_{seq} fitness measure and the candidate codes for the vertical coding model used the fit_{avg} fitness measure.

- Scaled Fitness - The raw fitness value was scaled using $C_m = 2$ and previously described techniques (see Chapter 4). Although the use of scaled fitness values for selection was explored, there was not any indication that the use of scaled fitness values increased GA performance.

- Target Sampling Rate - The target sampling rate was calculated using the raw fitness values. An individual's tsr was calculated using Equation 5.8:

$$tsr_i = 1 + \frac{f_i}{f_{avg}} \qquad (5.8)$$

  where f_i was the individual's fitness and f_{avg} was the average fitness for the generation. Individual tsr values were used to select reproductive partners.

Genetic Operators Used on Individuals

The three genetic operators used to move the population towards the optimal solution were selection, crossover and mutation.

- Selection - The binary GA used target sampling rate selection to determine the number of times an individual would be allowed to mate. The higher the fitness of an individual, the larger their tsr value, hence, the more offsprings they produced. Once the tsr values of candidate codes were calculated, selection of mating pairs proceeded as described in Chapter 4.
- Crossover - Similar to the GA for base five codes, this GA uses parameterized uniform crossover to produce new individuals from mating pairs. The crossover rate is set to $P_C = 0.83$ and the crossover probability for an individual locus is $P_{locus} = 0.5$. Given the crossover probabilities, children are produced as described in Chapter 4, Section 4.2.3.
- Mutation - After new individuals were produced using the crossover operator, each locus underwent mutation at a rate of $P_M = 0.0667$, which was calculated using Equation 5.9:

$$P_M = \frac{1}{n * L} \tag{5.9}$$

  where $n * L$ is the length of an individual's binary chromosome [7]. Since the length of an individual was less than the length of an individual for base five codes, the mutation rate was increased to improve performance and avoid premature convergence. The binary GA also used elitism to ensure the elite individual in the present generation was as fit or more fit than the elite individual of the previous generation. This prevented the best solution from being lost as the generations progressed.

Motif-Based Fitness

As previously discussed, the f_{seq} fitness measure assumes all positions in the messenger RNA are equally important for ribosomal binding hence for initiation. But the use of motif-based fitness measures allow positions which have been shown to be more biologically relevant to have increased fitness weight values. The statistical analysis of binding relationships between mRNA leader sequence and the exposed portion of the 16S rRNA corroborates what has been reported in literature [8]: alignment of the 16S rRNA at specific positions on the leader of mRNA is more favorable than at other positions. Motif-based fitness was calculated using Equation 5.10:

$$f_{seq}^{motif} = 1 - \frac{1}{NumValid * nGSHIFTS}\left(\sum_{i=1}^{NumValid*nGSHIFTS} Sdist_i * Mvec_i\right) \tag{5.10}$$

where, $Mvec$ is the one by $nGSHIFTS$ motif vector of positional weight values between 0 and $nGSHIFTS$. Again, for accurate fitness calculation,

$$\sum_{i=1}^{NumValid*nGSHIFTS} Mvec_i = NumValid * nGSHIFTS$$

The motif weight vector used for binary horizontal codes was designed such that position -18 to -9 contained eight-six percent of the weight. Table 5.3 shows the weighting distribution. The distribution was based on the hydrogen-binding analysis results (shown in Figure 5.2) for regions in the sixty-base mRNA leader sequence. Fitness values for motif-based codes were higher than for equal-weight codes since a zero at a key position could significantly inflate the fitness value.

Directional Codes

As previously discussed, coding can occur horizontally, over a single leader sequence and possibly vertically, over specific positions in the mRNA leader. This is especially true when

Table 5.3: Regional weighting distribution for motif-based fitness calculation

Gmask Alignment Position in mRNA	Approximate Weighting Percentage
-30 to -19	4
-18 to -16	18
-15 to -13	45
-12 to -10	12
-9 to -7	11
-6 to -4	2
-3 to -1	2
0 to +2	2
+3 to +5	2
+6 to + 17	2

modeling the functional behavior of the translation initiation system. Since binding analysis already indicated the importance of positional behavior in translation initiation, it was a logical step to investigate horizontal and vertical code models to describe the interaction between the ribosome and the mRNA leader region.

Horizontal Codes: As stated in earlier chapters, the horizontal code models assumed that the binding pattern found in regions of the mRNA leader could be modeled using a single code. Therefore one gmask set could be applied to all binding patterns in the mRNA. To search for the best $(n = 3, k = 1, m = 4)$ horizontal code, candidate codes' gmasks were applied to the binding patterns of mRNA leader sequences. There were $NumValid * nGSHIFTS = 96$ possible zero positions for a given mRNA leader. The GA found the best horizontal code model for each leader sequence based on the f_{seq} and f_{seq}^{motif} fitness measures.

Vertical Codes: Vertical codes described the positional behavior of the entire training set of leader sequences. The vertical code model assumed that positional similarities exist within the binding patterns describing mRNA/ribosomal interaction. The existence of vertical codes could imply the existence of positional error-control codes for protecting vital information bearing regions in mRNA leaders. Candidate vertical codes' associated gmasks were applied to all $nSEQ = 266$ leader sequences in the training set and the fitness was evaluated using the fit_{avg} fitness measure.

5.2.2 Procedure

Binary coding operations were written as a library of error-control coding operations, *codeOpsBin.c*. The genetic algorithm program, *gaOpsBin.c*, used the binary table-based code library to implement the horizontal and vertical code searches.

Messenger RNA leader sequences from *E. coli* K-12 genome were used as training sequences for finding the best candidate code model. The *E. coli* mRNA sequence set used in this work was the same set parsed (from the original GenBank files) for verifiable translation and used by Rosnick [65] and in Chapter 4 of this work. There were 531 sequences in the Rosnick sequence set. Every even sequence (where the first sequence in the file was considered sequence zero) was designated as part of the training set. There were 266 *E. coli* leader sequences in the training set used to evaluate candidate codes.

The GA was executed $RUNS\_PER\_SEQ$ times. Multiple runs were required to ensure adequate exploration of the search space starting from various solution points. Since the initial population was randomly generated, multiple runs should locate several possible optima codes. The number of runs per sequence/position was set based on the total run time of the GA and the size of the solution space. The GA was implemented and executed using a 600MHz Microsoft Windows 2000 machine. The following parameters were set for the horizontal and vertical code GAs:

- Horizontal Table-Based Codes - The GA searched for horizontal codes that best described each of the 266 leader sequences in the training set. For the horizontal code GA,

  $$POPULATION\_SIZE = 1000$$

  $$NUMBER\_GENERARIONS = 500$$

  $$RUNS\_PER\_SEQ = 10$$

  The total time to execute the GA for all 266 sequences was approximately 73 hours. The motif-based horizontal GA used the same parameters and executed in about the same amount of time. As previously mentioned, the search space for the (3,1,4) binary code model is small compared to the search space of the (3,1,4) base five code model and an exhaustive search would have been sufficient. Also, a GA with smaller population size and generations would have been quicker and more efficient.

- Vertical Table-Based Codes - The GA searched for vertical codes for each of the possible $NumValid = 48$ positions on the sixty-base leader sequence. The fitness was based on the average fitness over all of the 266 *E. coli* leader sequences in the training set. For the vertical code GA,

  $$POPULATION\_SIZE = 500$$

  $$NUMBER\_GENERARIONS = 50$$

  $$RUNS\_PER\_SEQ = 10$$

  Since parameters for the vertical GA were smaller than the horizontal GA the execution time was approximately 3 hours. Use of a GA proved an efficient search technique.

Table 5.4: Contents of summary data file

Column	**Data**
1	Sequence identity (all even numerals) or Position Number (Vertical Codes)
2	Run Number - which run (out of $RUNS\_PER\_SEQ$) the best code occurred
3	Generation Number - which generation best code occurred; always equal to $NUMBER\_GENERATIONS$ since GA uses elitism
4	Maximum Fitness - the fitness of the best code; also the maximum fitness for the generation
5	Minimum Fitness - the lowest fitness for the generation
6	Average Fitness - the average fitness for the generation
7 to 11	Coefficients for generator 1, C_1, of the best code
12 to 16	Coefficients for generator 2, C_2, of the best code
17 to 21	Coefficients for generator 3, C_3, of the best code
22 to 30	Coefficients for the nine-base gmask1 for the best code
31 to 39	Coefficients for the nine-base gmask2 for the best code
40 to 135(horiz)/41(vert)	Syndrome distance pattern for the best code

The genetic algorithm search program produced three output files:

- Summary File - A Matlab formatted space delimited data file. Each row of the summary file contained information for the best code candidate (based on individual sequence fitness) out of all possible runs. For the horizontal codes the summary data file contained 266 rows and for vertical codes the summary data file contained 48 rows.

  For a given row, Table 5.4 defines the information stored in each column:

- All File - A LOG (text) comma delimited file containing the same information as the summary file (see Table 4.5) for the best code in each run. The all log file contained $RUNS\_PER\_SEQ * 266$ rows for the horizontal GA and $RUNS\_PER\_SEQ * 48$ for the vertical GA. In the all file, the coefficients of the

Table 5.5: Contents of the all log file

Column	**Data**
1	Sequence identity (all even numerals) or Position Number (Vertical Codes)
2	Run Number
3	Generation Number - which generation best code occurred; always equal to $NUMBER\_GENERATIONS$ since GA uses elitism
4	Maximum Fitness - the fitness of the best code; also the maximum fitness for the generation
5	Minimum Fitness - the lowest fitness for the generation
6	Average Fitness - the average fitness for the generation
7	C_1 of the best code
8	C_2 of the best code
9	C_3 of the best code
10	Gmask for the best code
11	Syndrome distance pattern for the best code

generators, gmasks, and syndrome distance were written as one string. Table 5.5 defines the columns of the all file:

- Population File - A LOG file that stored the details of the GA run including: decimal binding representation of the leader sequences in the input file, begin and end time for GA run, GA parameters, fitness motif weight vector, and detailed information about individual codes with fitness values above the fitness threshold score (45% for non-motif horizontal codes, 75.5% for motif-based horizontal codes, and 55% for vertical codes).

The output of the GAs were analyzed; results are presented and discussed in the following section.

5.2.3 Results

The summary data files for the horizontal, motif-based horizontal, and vertical binary table-based convolutional code models were analyzed using routines developed with *Matlab*.

As in the base five case, the syndrome distance vectors of the binary code models indicate how well the code recognizes the subsequence being decoded. An all-zero syndrome distance vector would have a fitness value of one, indicating that the sequence being analyzed was produced by the code model. Figure 5.3 shows the average syndrome distance value for the optimal codes discovered using the *E. coli* model set. In Figure 5.3 the horizontal axis is

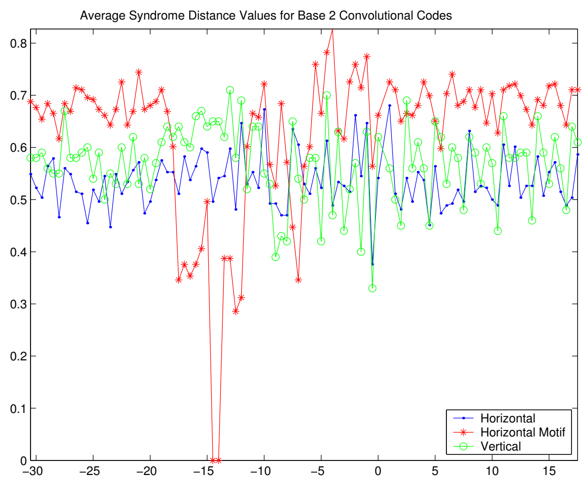


Figure 5.3: Average Syndrome Distance From the All-Zero Syndrome for Binary Table-Based Convolutional Code Models for Translation Initiation

position relative to the first base in the initiation codon and the vertical axis is the average syndrome distance value.

The performance of the binary vertical code is better, based on average syndrome distance analysis in Figure 5.3, than the base five vertical code models. The vertical code's mean syndrome distance value are comparable to those of the horizontal models, especially the equal-weight horizontal model. Both the equal-weight horizontal model and the vertical model reach their minimum average syndrome distance values around position 0. The motif-based horizontal code model achieves a zero average syndrome distance at position -14, which is the position with the greatest binding difference between leader regions and non-leader intergenic sequences. The motif-based horizontal model achieves most of its minimum distance values between -18 and -12 and again at -7. The equal-weight horizontal model appears to perform better in the region preceeding -18, but this may be a result of a higher number of binding patterns with low hydrogen binding. All-zero or heavily-zero binding vectors, the presence of zeros in the code model, and heavily zero gmask coefficients can lead to an inflated performance.

The gmask coefficients were analyzed to determine which binding regions and to what degree binding relationships between the mRNA leader sequence training set and the exposed portion of the 16S rRNA were captured by the code models. For each twelve bit binary binding sub-pattern, the nine bit gmask shifts twice. Each shift corresponds to binding with a different region of the last thirteen bases of the 16S rRNA:

$$Shift_1 = (\ldots\ \ A\ \ U\ \ U\ \ C\ \ C\ \ U\ \ C\ \ C\ \ A\ \ ..)$$

$$Shift_2 = (\ldots\ \ C\ \ C\ \ U\ \ C\ \ C\ \ A\ \ C\ \ U\ \ A\ \ ..)$$

For the nine bit gmask, coefficient values of one indicate a position on shift sequence one or two with which the mRNA leader must form a hydrogen bond. Figure 5.4 shows the average gmask values for the code models. The horizontal axis indicates bit position in the gmask vector. The vertical axis is the average value over all codes in the model set. Position seven on both gmasks and, to a slightly lesser degree, position four are the results of the gmask construction method used in table-based codes. Figure 5.4 indicates that the gmasks for vertical codes contain a relatively large number of zeros in many positions. Large number of zeros in the gmask of the vertical code model probably inflated the syndrome distance performance results of the vertical code model. Although the equal-weight horizontal code's

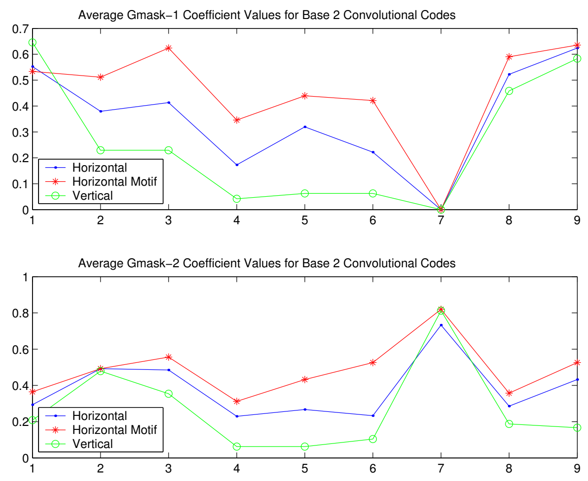


Figure 5.4: Average Gmasks Values for Binary Table-Based Convolutional Code Models for Translation Initiation

gmasks contained fewer zeros than the vertical code, its gmasks still contained more zeros than the motif-based horizontal code's gmasks. For gmask1, position one to three had relatively high average coefficient values for the motif-based codes. This corresponds to the first three codons in shift one or shift two: (A U U) or (C C U). The last two positions of gmask2 (motif-based model) also indicated high binding, corresponding to binding with the last two bases in shift one or two: (C A) or (U A). The high binding areas for gmask2 of the motif-based horizontal code model is positions two and three and position six and seven (although position seven is a result of the method used), corresponding to (U U) and (U C) or (C U) and (A C).

5.2.4 Discussion

The binary GA models performed comparable to one another, the motif-based method captured the functional behavior of the ribosome binding site better than the other two models. Unlike the base five models where all-zero parity sequences do not occur, the binary code models are affected by all-zero binding patterns. The affects of the all-zero parity can be minimized by using motif-based fitness measures over regions in the mRNA with greater binding affinity to the exposed portions of the 16S rRNA. Use of fitness penalties for all-zero parity sequences may improve the models.

The resulting gmasks were affected, as in the base five case, by the table-based gmask construction method. Investigating other decoding methods and increasing the memory length of the code should improve the resulting models. Functional code models for protein translation initiation aid in understanding the system and can help define the binding behavior that is necessary for translation initiation. This can lead to improvements in the sequence based coding models and to algorithms for designing and improving the efficiency of transgenic leader sequences. Functional code models can be extended to incorporate more specific binding information such as the number of hydrogen bonds formed. Results of the binary binding pattern are encouraging.

In the following chapter, the base five and the binary table-based convolutional code models are tested against prokaryotic leader sequences and *E. coli* non-leader intergenic sequences.

Chapter 6

Evaluation of Coding Models for Prokaryotic Translation Initiation

The 266 horizontal table-based base five and horizontal table-based binary coding models produced by the genetic algorithm search technique were tested on data sets from four prokaryotic organisms: *E. coli* K-12, *S. typhimurium* LT2, *B. subtilis*, and *S. aureus* Mu50. *E. coli* and *S. typhimurium* share a common lineage as do *B. subtilis* and *S. aureus*. In addition to the test run on prokaryotic leader regions, *E. coli* K-12 intergenic, non-leader regions were tested. The sequence data sets used for testing the model on prokaryotic organisms were compiled and processed using the web-based GenBank Information Retrieval Tool developed by Cheng and Chandra of the North Carolina State University Scientific Data Management Center. Using *modelTest.c* (base five model testing) and *modelTestBinary.c* (binary model testing), each test data set was decoded using the gmasks of the 266 code models. The fitness of each model was calculated, as described in previous chapters, based on the resulting syndrome vector for each individual sequence in the data set. The performance result for the first model with the highest fitness score or the elite model was recorded in the summary data file for the test data set. Performance results for models with fitness scores equal to the fitness score of the elite model were recorded in the all data file. The population log file stored details of the test, including detailed information of any

Table 6.1: Contents of test data sets' summary and all data file

Column	**Data**
1	Sequence identity of test sequence
2	Index of the elite code model (0 to 265)
3	Sequence identity of *E. coli* sequence used to develop elite code model
4	Maximum Fitness - the fitness of the elite code model
5	Minimum Fitness - the lowest fitness for the set of all code models
6	Average Fitness - the average fitness for the set of all code models
7 to 24(base 5)/102(binary)	Syndrome distance pattern associated with the elite code model

code model/test sequence pair with fitness statistics above fitness thresholds used during model construction. The contents of the summary and all data files are described below in Table 6.1.

The test results stored in the summary and all data files were analyzed for each data set using analysis routines developed with *Matlab*. The results from Chapters 4 and 5 showed that it is possible to construct table-based, convolutional codes which have decoding masks that resemble the 3' end of the 16s rRNA and which recognize key initiation regions in mRNA leader sequences. If the code sets constructed in preceding chapters are valid models for initiation, then they should recognize the *E. coli* testing set and fail to recognize non-leader sequences (indicate that non-leader sequences are errors). It is expected that, if the $(3, 1, 4)$ code models depict translation initiation accurately, then there will be notable regions of distinction between the leader and non-leader sequences, particularly in the non-random (-20 to -13) and Shine-Dalgarno (-10) domains. Also, it is expected that the current convolutional code models will respond differently to prokaryotic organisms of varying taxonomical relatedness to *E. coli*, having similar analysis results for related organisms and different analysis results for distantly related organisms. The results for each test data set tested using the base five and binary code models are presented and discussed in the sections which follow.

6.1 Base Five Table-Based Code Models

The table-based base five code models were applied first to the *E. coli* leader sequence and non-leader intergenic sequence test sets to see if the model performed differently for the two data sets. The model was also applied to prokaryotic organisms to discover how the model reacts to organisms close in lineage to the model organism and to prokaryotic organisms distantly related to the model organism.

6.1.1 *E. coli* K-12

Equal-Weight Horizontal Codes

Figure 6.1 shows the average syndrome distance value for the equal-weight base five codes tested against translated and non-translated *E. coli* test set. In Figure 6.1 the horizontal axis is position relative to the first base in the initiation codon and the vertical axis is the average syndrome distance value. Ideally, there would be a drop in the syndrome distance of the leader regions in the non-random and the Shine-Dalgarno domain but there would not be a drop in the syndrome distance of non-leader sequences. The performance of the *E. coli* test sets is not as good as the model, but there is some evidence of differing average behavior between the *E. coli* leader and non-leader test sets. The *E. coli* leader sequence test set achieves its minimum average syndrome distance value at position -15 (within the non-random domain) while the non-leader intergenic sequences achieve their minimum average

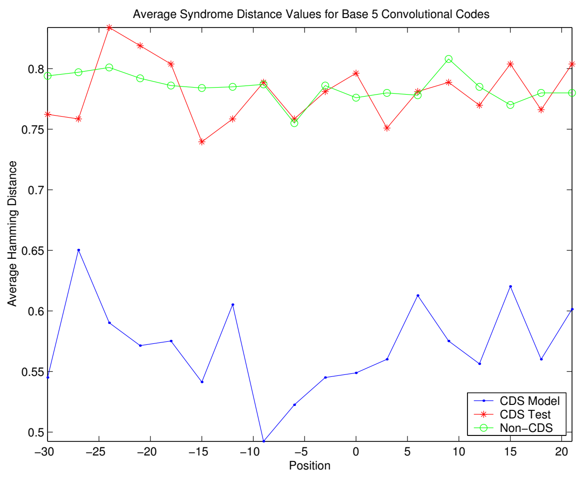


Figure 6.1: Average Syndrome Distance for *E. coli* Test Set Tested with Base Five Code Models

syndrome distance value at position -6. This low syndrome distance value may be due to the presence of the AUG codon since both groups have a local minima at this position.

Syndrome distance is directly related to the fitness of a code or code/sequence pair. The number of zeros in the syndrome vector determine both the syndrome distance and the fitness. Table 6.2 shows the correspondence between fitness value and number of zeros in the syndrome vector for the equal-weight horizontal code model.

Figure 6.2 shows the fitness distribution for the equal-weight base five codes tested against translated and non-translated *E. coli* test set and the fitness distribution for the model. In Figure 6.2 the horizontal axis is the fitness values and the vertical axis is the probability of each value occurring. As expected, based on the average syndrome distance results, fitness values for the test sets are lower than the fitness values of the model. The fitness distribution for the leader set and the non-leader set are very similar. While most of the training models' fitness is around 0.444 (corresponding to eight zeros in the syndrome vector), most of the test sets' fitness is around 0.222 (four zeros in the syndrome distance). Although fitness does not indicate the location of zero syndromes, it was expected that the

Table 6.2: Relationship between number of zeros in syndrome vector and fitness value

Number of Zeros	Fitness
0	0.000
1	0.056
2	0.111
3	0.167
4	0.222
5	0.278
6	0.333
7	0.389
8	0.444
9	0.500
10	0.556
11	0.611
12	0.667
13	0.722
14	0.778
15	0.833
16	0.889
17	0.944
18	1.000

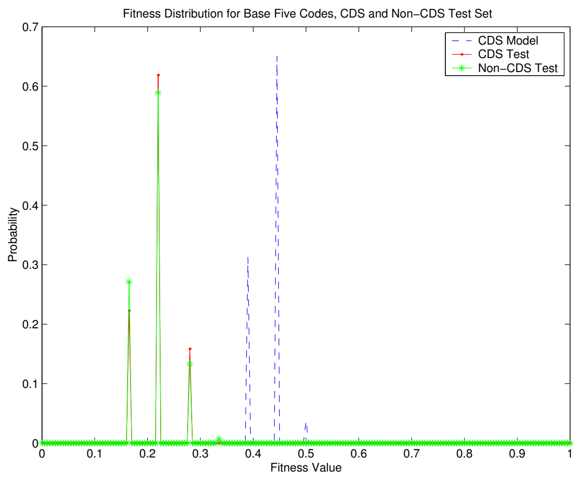


Figure 6.2: Fitness Distribution for *E. coli* Test Set and Base Five Code Models

leader set's fitness would be higher than the non-leader set. The leader set has a slightly higher probability of fitness values being in the 0.222 and 0.278 (five zeros) range than the non-leader set. The non-leader group has a slightly higher probability of fitness values in the 0.167 (three zeros)

Although the fitness distribution of the leader and non-leader sets are similar, their syndrome patterns differ, as indicated by Figure 6.1, implying that their code preferences may also differ. The usage ratio compares the number of times a code is selected as the most fit by a valid initiation sequence to the number of times the code is selected by an invalid initiation sequence. Figure 6.3 shows the usage ratio for each code in the base five coding model set. In Figure 6.3 the horizontal axis is the index representation for the codes

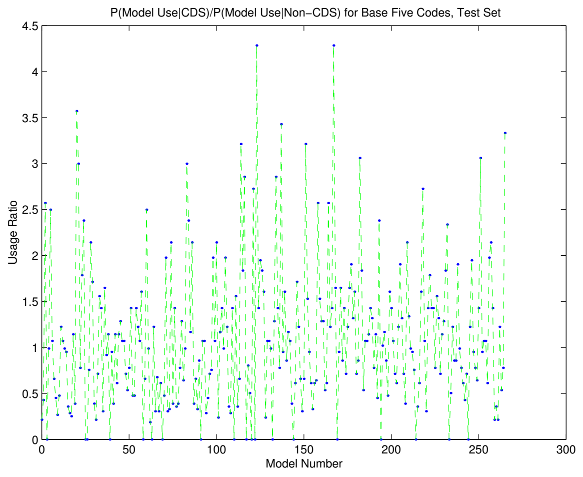


Figure 6.3: Base Five Code Usage Ratio for *E. coli* Test Set

in the model and the vertical axis is the corresponding usage ratio. The usage ratio shows a distinct preference by the leader and non-leader set for certain code models over other models. There are few codes with equal usage values; this indicates a distinction between the two test sets based on code model preference.

Motif-Based Horizontal Codes

Figure 6.4 shows the average syndrome distance value for the motif-based base five codes tested against translated and non-translated *E. coli* test set. In Figure 6.4 the horizontal

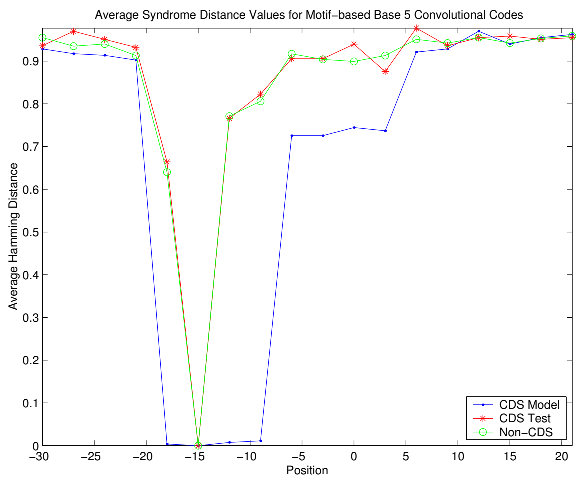


Figure 6.4: Average Syndrome Distance for *E. coli* Test Set Tested with Base Five Motif-Based Code Models

axis is position relative to the first base in the initiation codon and the vertical axis is the average syndrome distance value. The two test sets behave very similar to the model. Both haver zero average syndrome distance values at position -15 but neither has any other significantly low syndrome distance point elsewhere. The expectation was that the leader set would perform similar to the model in the non-random and Shine-Dalgarno domain and that the non-leader would not. But, there are no clear differences between the average syndrome distance for the leader set and the average syndrome distance for the non-leader set in the -15 to 0 region.

Figure 6.5 shows the fitness distribution for the equal-weight base five codes tested against translated and non-translated *E. coli* test set and the fitness distribution for the model. In Figure 6.5 the horizontal axis is the fitness values and the vertical axis is the

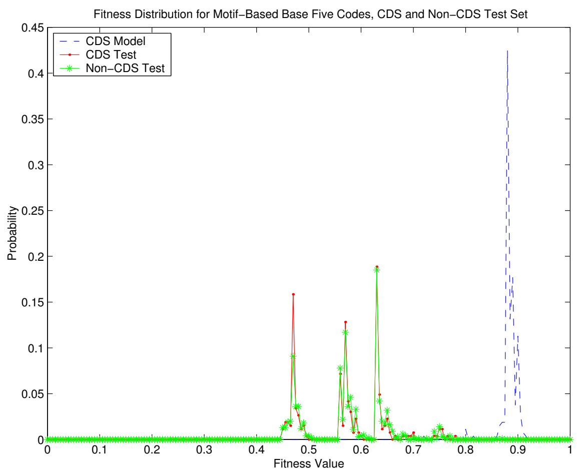


Figure 6.5: Fitness Distribution for *E. coli* Test Set and Base Five Motif-Based Code Models

probability of each value occurring. The fitness distribution of the test sets are centered over lower fitness regions than the model. The fitness values of the test sets are over three main fitness regions (around 0.475, 0.575, and 0.625) while the model is centered over a single, higher valued, fitness region (around 0.9). Ideally the leader set would contain a larger percent of sequences in the high fitness region and the non-leader would have a larger percent of its sequences in the lower fitness regions. There are differences between the fitness distribution of the leader and the non-leader sets. The leader test contains more fitness values over the first fitness region and slightly more fitness values in the second fitness region. The two test sets contains about the same fitness values in the last fitness region.

Figure 6.6 shows the usage ratio for each code in the base five coding model set. In Figure 6.6 the horizontal axis is the index representation for the codes in the model and the vertical axis is the corresponding usage ratio. There are greater usage differences in leader and non-leader test set for the motif-based model than the equal-weight model, with the highest usage ratio reaching ~ 11.75. The results of the usage ratio analysis again

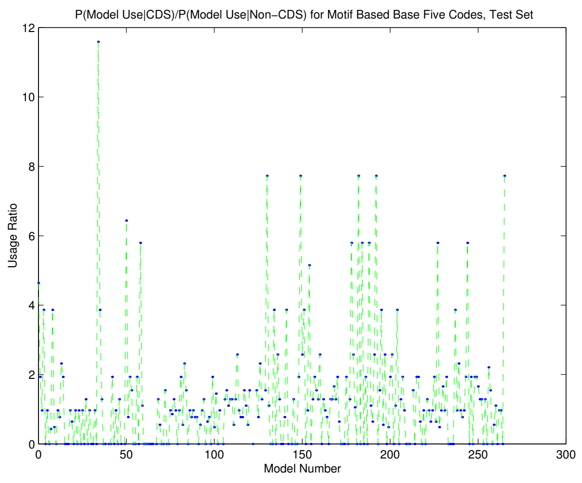


Figure 6.6: Base Five Motif-Based Code Usage Ratio for *E. coli* Test Set

demonstrate different code model preferences between the leader and non-leader test set.

6.1.2 Application to Other Prokaryotic Organisms: *S. typhimurium* LT2, *B. subtilis*, and *S. aureus* Mu50

The base five horizontal code models were applied to other prokaryotic test sets: *S. typhimurium*, a close relative of *E. coli*, *B. subtilis* and *S. aureus*, distant relatives of *E. coli*.

Equal-Weight Horizontal Codes

Figure 6.7 shows the average syndrome distance value for the equal-weight base five codes tested against the prokaryotic test set. In Figure 6.7 the horizontal axis is position relative to the first base in the initiation codon and the vertical axis is the average syndrome distance value. If the model distinguishes between close and distant organisms, then *B. subtilis* and *S. aureus* would be expected to have similar syndrome distance patterns and for these patterns to differ from those of *S. typhimurium*. All three prokaryotes are expected to have overall behavior similar to the model in the non-random and Shine-Dalgarno domains, with

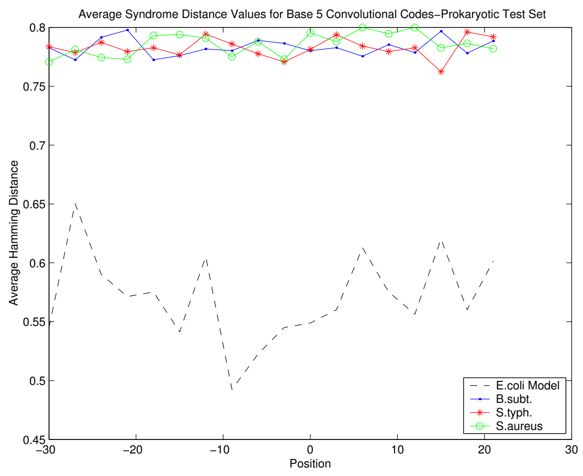


Figure 6.7: Average Syndrome Distance for Prokaryotic Test Set Tested with Base Five Code Models

S. typhimurium having the lowest syndrome distance values since it is taxonomically closer to *E. coli.* The average syndrome distance of all three sets is higher than the model and in some areas higher than the *E. coli* leader and non-leader test sets. Although *S. aureus* has slightly higher average syndrome distance values in positions -18, -15, and from +6 to +12, there are no clear behavioral differences in the syndrome distance values of the three prokaryotic test organisms.

Figure 6.8 shows the fitness distribution for the equal-weight base five codes tested against prokaryotic test set and the fitness distribution for the model. In Figure 6.8 the horizontal axis is the fitness values and the vertical axis is the probability of each value occurring. Similar to the *E. coli* test sets, the most probable fitness value is 0.222 (four zeros) and the distributions of the prokaryotic test sequences are similar to the model but in lower fitness regions. There are minimal differences between the fitness distribution of the three prokaryotic test sets.

The usage ratio for the prokaryotic test organisms compares the number of times a code is selected as the most fit by an *E. coli* initiation test sequence to the number of times the

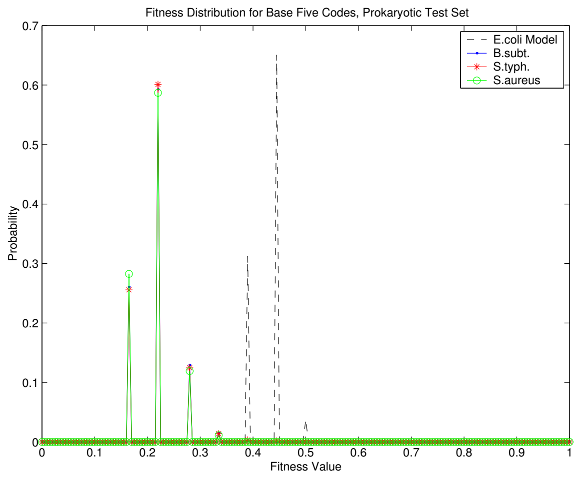


Figure 6.8: Fitness Distribution for Prokaryotic Test Set and Base Five Code Models

code is selected by the other prokaryotic initiation sequences. Figure 6.9 shows the usage ratio for the *B. subtilis*. Figure 6.10 shows the usage ratio for the *S. typhimurium* LT2.

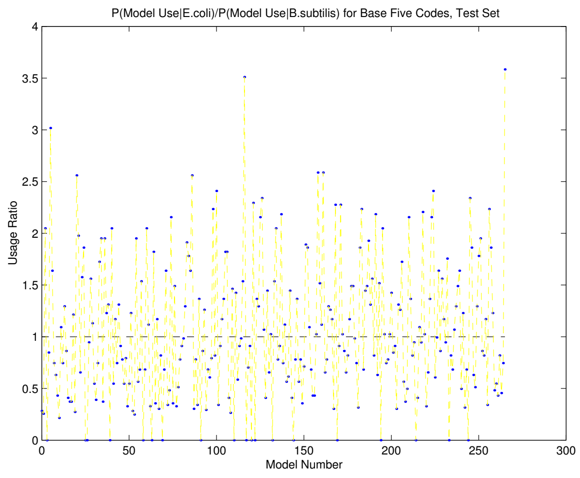


Figure 6.9: Base Five Code Usage Ratio for *B. subtilis* Test Set

Figure 6.11 shows the usage ratio for the *S. aureus* Mu50. In Figure 6.9, Figure 6.10, and Figure 6.11 the horizontal axis is the index representation for the codes in the model and the vertical axis is the corresponding usage ratio. There are higher usage ratio values for *E. coli* leader versus *E. coli* non-leader than for *E. coli* leader versus prokaryotic leader. The

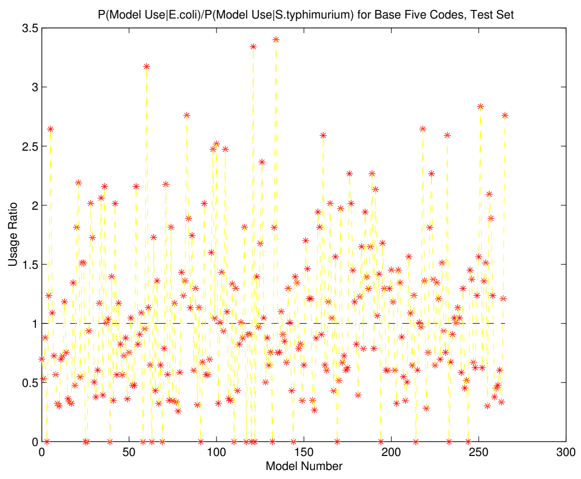


Figure 6.10: Base Five Code Usage Ratio for *S. typhimurium* Test Set

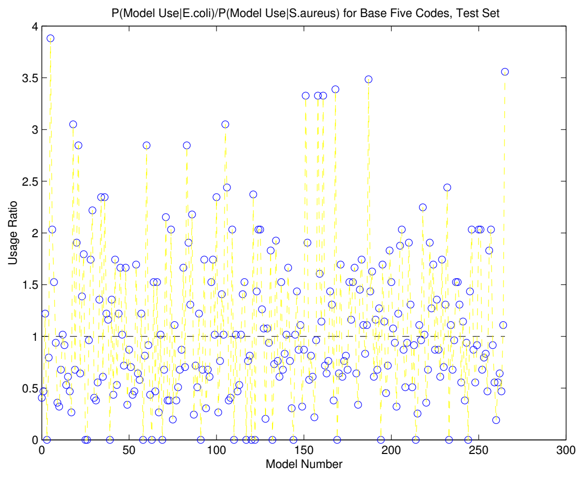


Figure 6.11: Base Five Code Usage Ratio for *S. aureus* Test Set

usage ratios are different for each prokaryotic model, implying that different code models are preferred by different organisms.

Motif-Based Horizontal Codes

Figure 6.12 shows the average syndrome distance value for the motif-based, base five codes tested against the prokaryotic test set. In Figure 6.12 the horizontal axis is position relative

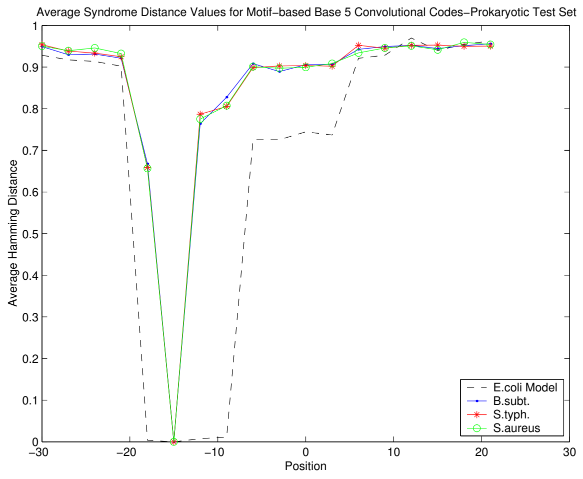


Figure 6.12: Average Syndrome Distance for Prokaryotic Test Set Tested with Motif-Based Base Five Code Models

to the first base in the initiation codon and the vertical axis is the average syndrome distance value. It was expected that the closer relative of the model organism, *S. typhimurium*, would have lower syndrome distance values than the distant relatives of *E. coli*. But the general behavior of the prokaryotic test set is expected to be similar to that of the model in the non-random and Shine-Dalgarno regions. As in the *E. coli* test sets, all three prokaryotic test sets have a zero average distance value at position -15, the position with the highest weight or the greatest error protection. The organisms' average syndrome distance values are highly similar in each position of the leader sequence. Although slight, there is more of a difference between the *E. coli* leader and non-leader test sets than between the prokaryotic leader test sets.

Figure 6.13 shows the fitness distribution for the motif-based, base five codes tested against prokaryotic test set and the fitness distribution for the model. In Figure 6.13

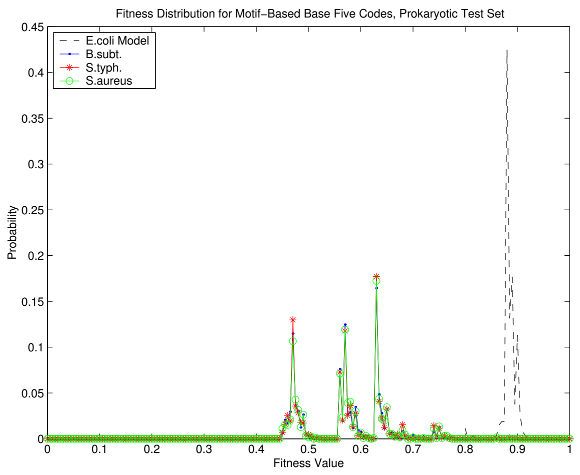


Figure 6.13: Fitness Distribution for Prokaryotic Test Set and Motif-Based Base Five Code Models

the horizontal axis is the fitness values and the vertical axis is the probability of each value occurring. The fitness distribution for the prokaryotic test is centered over the same three fitness values that the *E. coli* test sets were centered over. The fitness values of the prokaryotic organisms are less than the fitness values for the model. Similar to the *E. coli* leader test set, *S. typhimurium* has slightly more fitness values in the ~ 0.475 fitness region than the other two prokaryote test sets.

The usage ratio for the prokaryotic test organisms compares the number of times a code is selected as the most fit by an *E. coli* initiation sequence to the number of times the code is selected by the other prokaryotic initiation sequences. Figure 6.14 shows the usage ratio for the *B. subtilis*. Figure 6.15 shows the usage ratio for the *S. typhimurium* LT2. Figure 6.16 shows the usage ratio for the *S. aureus* Mu50. In Figure 6.14, Figure 6.15, and Figure 6.16 the horizontal axis is the index representation for the codes in the model and the vertical

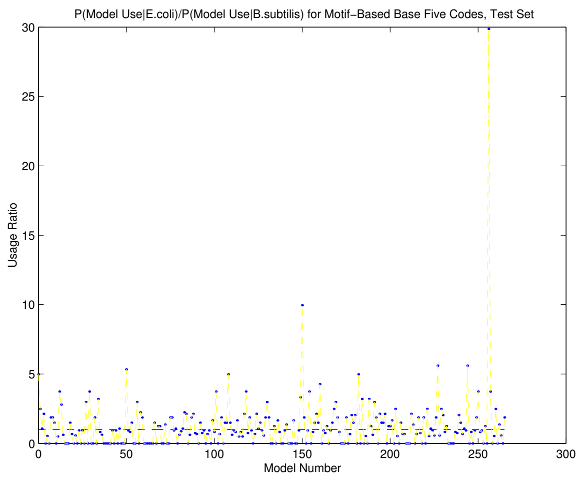


Figure 6.14: Motif-Based Base Five Code Usage Ratio for *B. subtilis* Test Set

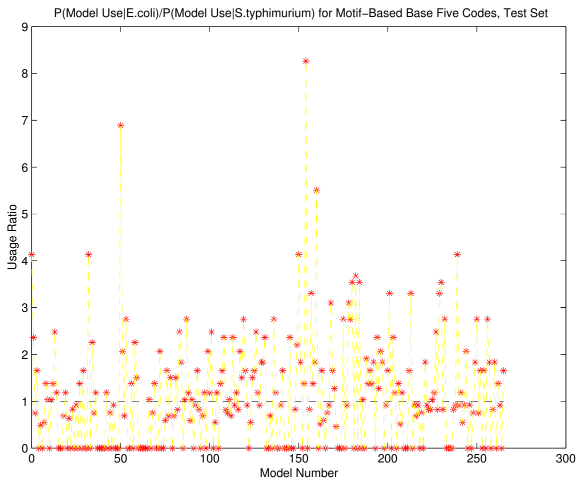


Figure 6.15: Motif-Based Base Five Code Usage Ratio for *S. typhimurium* Test Set

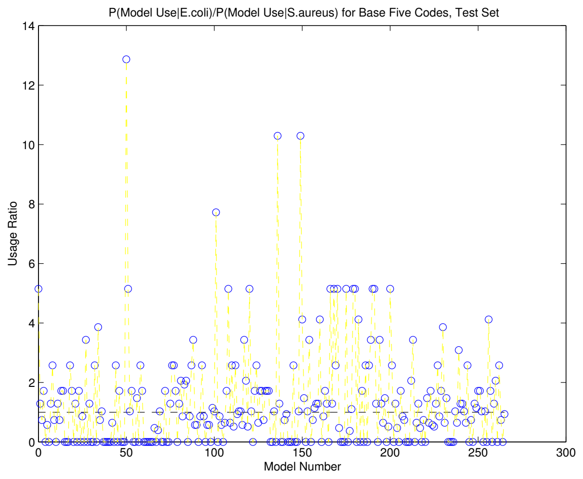


Figure 6.16: Motif-Based Base Five Code Usage Ratio for *S. aureus* Test Set

axis is the corresponding usage ratio. The highest usage ratio, ∼ 30, occurs for *B. subtilis* followed by *S. aureus* (∼ 12.5) and *S. typhimurium* (∼ 8.5) whose usage ratio is less than the highest usage ratio value of *E. coli* non-leader test sequences.

6.1.3 Discussion

Based on the average syndrome distance analysis, the equal-weight, base-five codes had more distinguishable positions than the motif-based code models for the leader and non-leader *E. coli* test sets. Given the current base five code models, equal-weight codes or equal error protection code models provide better error-detection when distinguishing leader and non-leader sequences. Motif-based codes can be viewed as codes with unequal error protection (UEP). UEP codes provide a higher error protection for some information symbols than for other information symbols. It follows that errors in symbols less protected will be more difficult to detect. Position -15 is the most protected position for the motif-based codes. Perhaps distributing the fitness weights more evenly over the -18 to -9 region would produce motif-based, base five models that distinguish between leader and non-leader sequences better than the current models.

Although the syndrome distance results do not show great distinction between the prokaryotic test sets and the *E. coli* leader and non-leader test sets, the usage ratio shows a distinct difference in model preference between the *E. coli* leader test set and the non-leader and prokaryotic test sets. Overall, usage ratio values were higher for motif-based codes than for equal-weight codes. Results of the usage ratio analysis indicate that it is highly probable to define a set of base five codes or systems that can distinguish between leader and non-leader parity sequences and also between self and non-self leader sequences. The usage ratio analysis can be used to refine the existing model to yield a basis set of codes with better error detecting ability.

6.2 Binary Table-Based Convolutional Code Models

Similar to the testing procedures used for the base five code models, the binary code models were also tested and results compared for *E. coli* leader and non-leader sequences. Also, the model was tested on related and distant prokaryotic organisms. The binary code model tests, are testing the functional behavior of the non-leader and leader sequences in the test sets.

An additional set of tests, termed "zero penalty" tests, were performed for the binary code models. For zero penalty testing, the value of each parity subsequence is checked prior to the final assignment of syndrome values. If the parity subsequence is an all zero sequence, then, regardless of the resultant syndrome value, a syndrome value of one is assigned to that parity subsequence. Zero penalty testing was designed to penalize sequences that result in a syndrome value of zero because they are all zero sequences. This helps prevent regions with low affinity to the 16S rRNA from having high fitness values. Traditional test results and zero penalty test results are presented.

6.2.1 *E. coli* K-12

Equal-Weight Horizontal Codes

Figure 6.17 shows the average syndrome distance value for the equal-weight binary codes tested against translated and non-translated *E. coli* test set. Figure 6.18 shows the average

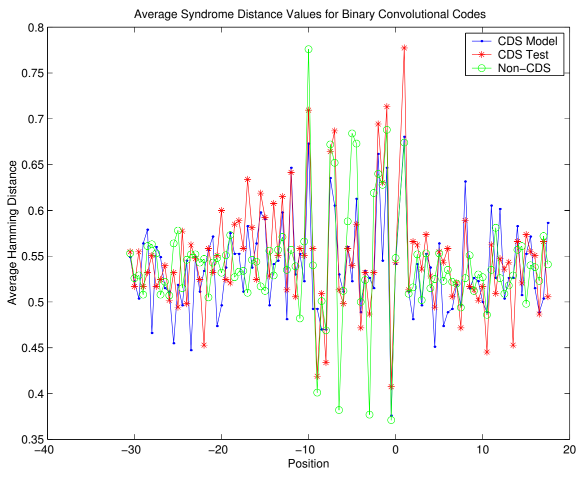


Figure 6.17: Average Syndrome Distance for *E. coli* Test Set Tested with Binary Code Models

syndrome distance value for the equal-weight binary codes tested against translated and non-translated *E. coli* test set, with penalties for zero. In Figure 6.17 and Figure 6.18 the horizontal axis is position relative to the first base in the initiation codon and the vertical axis is the average syndrome distance value. For Figure 6.17 and Figure 6.18, the lowest average minimum syndrome distance for the leader and the non-leader occurs at position 0. As in the base five models, the ideal model, consistent with the hypothesis of this work, would result in low syndrome distance values for the leader set within the non-random and Shine-Dalgarno domains and high (relative to leader test set) syndrome distance values in the non-leader set. The distance behavior of both test sets are comparable to that of the model but they differ from one another mostly in the region from -20 to -9.

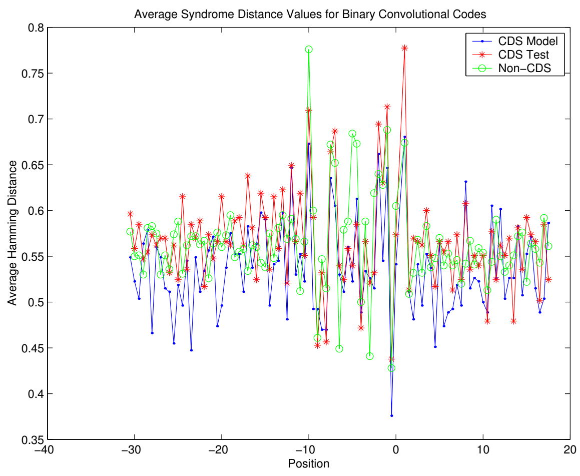


Figure 6.18: Average Syndrome Distance for *E. coli* Test Set Tested with Binary Code Models - Penalty for Zero

In this region, the leader test set's average syndrome distance values seem to be periodic in nature, oscillating between high distance values (higher than non-leader set) and low distance values. The non-leader has average distance values lower than or comparable to the leader test set in leader regions known to have affinity to the exposed part of the 16S rRNA. The low syndrome distance for non-leaders may be due to non-leader binary binding patterns with high number of zeros and code models with large number of zero coefficients. Figure 6.18 verifies the existence of the first case: non-leader binary binding patterns with high number of zeros resulting in low syndrome distance values. Comparing Figure 6.17 and Figure 6.18, a fitness penalty for all zero parity sequences increases the average syndrome distance (decreases fitness) for the non-leader set. This is especially evident in the region between -10 and 0.

Figure 6.19 shows the fitness distribution for the equal-weight binary codes tested against translated and non-translated *E. coli* test set and the fitness distribution for the model. Figure 6.20 shows the fitness distribution for the equal-weight binary codes tested against

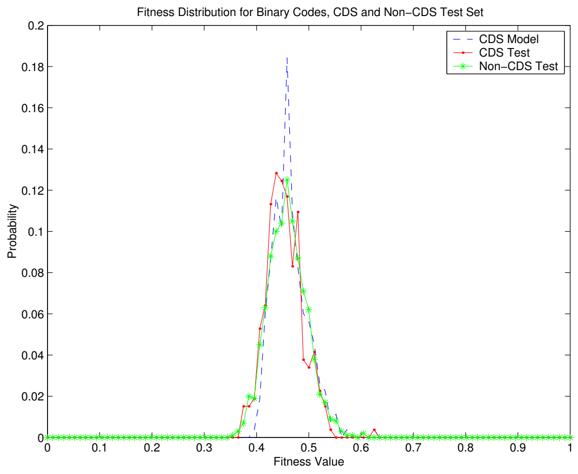


Figure 6.19: Fitness Distribution for *E. coli* Test Set and Binary Code Models

translated and non-translated *E. coli* test set, with penalties for zero, and the fitness distribution for the model. In Figure 6.19 and Figure 6.20 the horizontal axis is the fitness

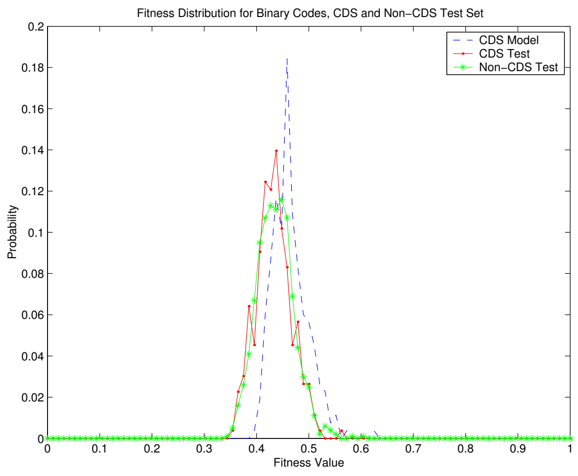


Figure 6.20: Fitness Distribution for *E. coli* Test Set and Binary Code Models - Penalty for Zero

values and the vertical axis is the probability of each value occurring. Ideally the fitness distribution of the leader set would resemble the model set while the non-leader set would notably differ from the leader set and the model set, with more sequences in the non-leader set having low fitness values. Unlike the equal-weight base five code models, the fitness distribution range is relatively the same for the the equal-weight binary model. The zero penalty analysis, Figure 6.20, reduces the fitness of the the *E. coli* test set and results in a slightly greater distinction between the leader and non-leader test sets. The fitness of the leader and non-leader are similar, with the highest probability occurring at a slightly lower fitness value for the *E. coli* leader set.

As in the base five analysis, the usage ratio compares the number of times a code is selected as the most fit by a valid initiation sequence to the number of times the code is selected by an invalid initiation sequence. Figure 6.21 shows the usage ratio for each code in the binary coding model set. Figure 6.22 shows the usage ratio for each code in

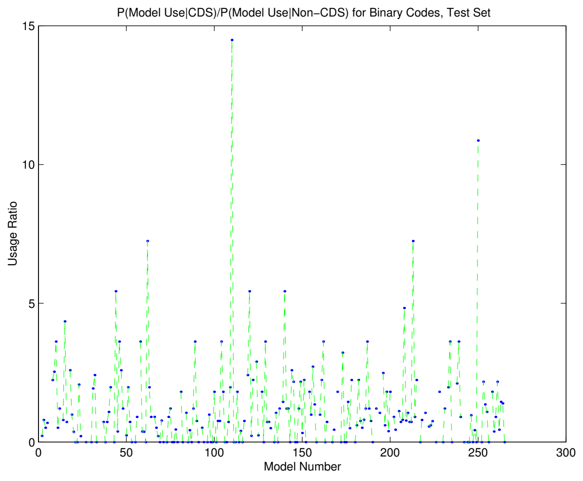


Figure 6.21: Binary Code Usage Ratio for *E. coli* Test Set

the binary coding model set, with penalties for zero. In Figure 6.21 and Figure 6.22 the horizontal axis is the index representation for the codes in the model and the vertical axis is the corresponding usage ratio. There are no notable differences between the usage ration for non-penalty and zero penalty tests. The maximum usage ratio values are significantly larger for the binary models than for base five code models, with the largest ratio being

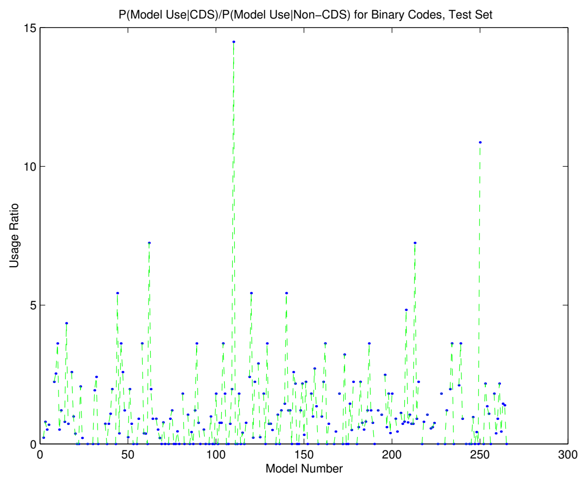


Figure 6.22: Binary Code Usage Ratio for *E. coli* Test Set - Penalty for Zero

$\sim$ 14.5 for binary and $\sim$ 4.25 for base five *E. coli* test sets.

Motif-Based Horizontal Codes

Figure 6.23 shows the average syndrome distance value for the motif-based binary codes tested against translated and non-translated *E. coli* test set. Figure 6.24 shows the average

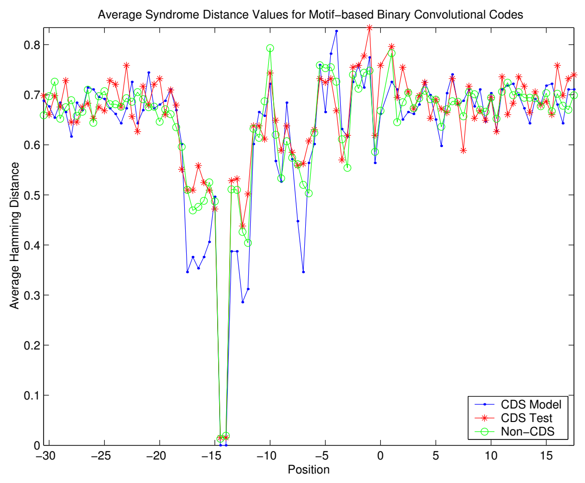


Figure 6.23: Average Syndrome Distance for *E. coli* Test Set Tested with Binary Motif-Based Code Models

syndrome distance value for the motif-based binary codes tested against translated and non-translated *E. coli* test set, with penalties for zero. In Figure 6.23 and Figure 6.23 the

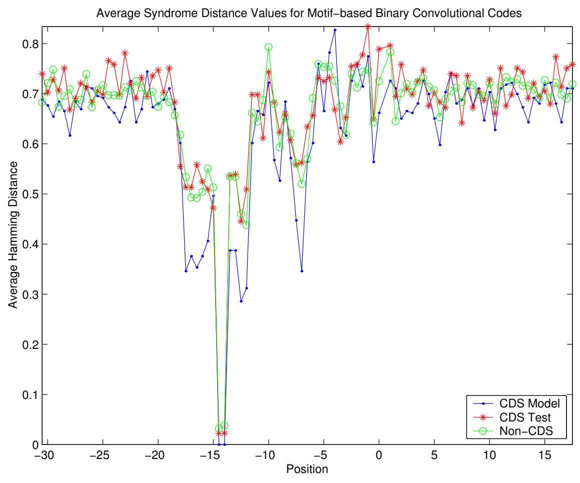


Figure 6.24: Average Syndrome Distance for *E. coli* Test Set Tested with Binary Motif-Based Code Models - Penalty for Zero

horizontal axis is position relative to the first base in the initiation codon and the vertical axis is the average syndrome distance value. Similar to previous models, the leader set was expected to have syndrome distance values lower than those of the non-leader set in the positions corresponding to the non-random and Shine-Dalgarno domains. Both test sets behave similar to the model, reaching their minimum distance values at position -14. Both shift positions at -14 carry the same weight. There are notable differences between the leader and non-leader sets and between the non-penalty and the zero penalty analysis, with the zero penalty analysis resulting in increased average syndrome distance values particularly for the non-leader test set.

Figure 6.25 shows the fitness distribution for the equal-weight binary codes tested against translated and non-translated *E. coli* test set and the fitness distribution for the model. Figure 6.26 shows the fitness distribution for the equal-weight binary codes tested against translated and non-translated *E. coli* test set and the fitness distribution for the model,

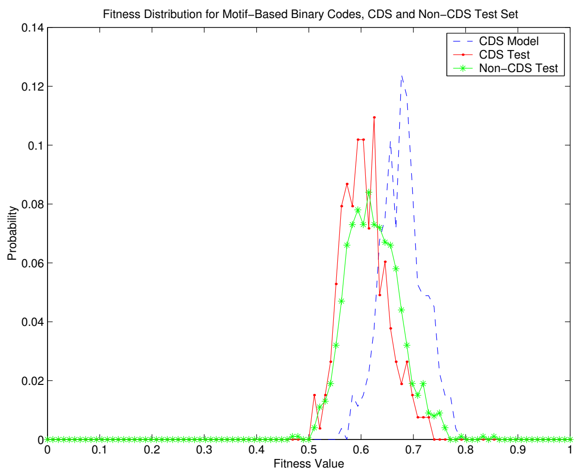


Figure 6.25: Fitness Distribution for *E. coli* Test Set and Binary Motif-Based Code Models

with penalties for zero. In Figure 6.25 and Figure 6.26 the horizontal axis is the fitness

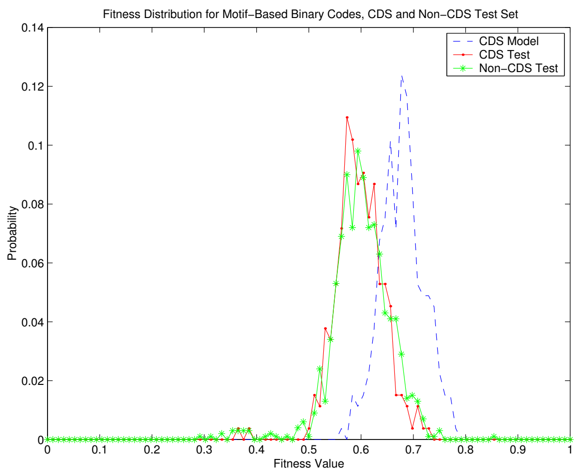


Figure 6.26: Fitness Distribution for *E. coli* Test Set and Binary Motif-Based Code Models - Penalty for Zero

values and the vertical axis is the probability of each value occurring. Similar to the equal-weight binary model, the zero penalty reduces the fitness values of the non-leader set. This is evident in Figure 6.26 where the distribution of the non-leader test set has less overlap with the model and more overlap with the leader test set. The leader set is less affected by the zero penalty, again verifying the view that the non-leaders positive performance is due to binary binding patterns with low affinity to the exposed portion of the 16S rRNA and high number of zeros.

Figure 6.27 shows the usage ratio for each code in the binary coding model set. Figure 6.28

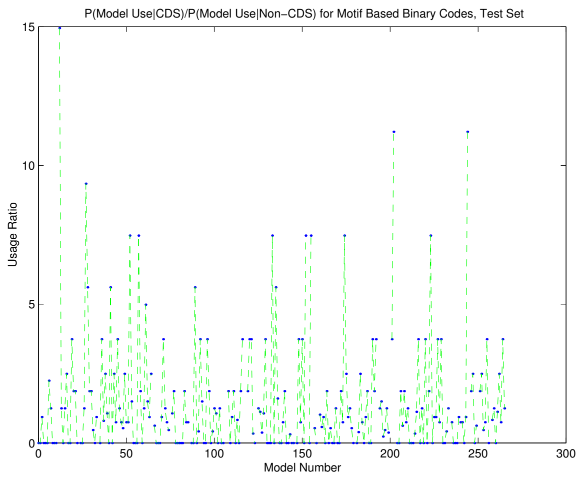


Figure 6.27: Binary Motif-Based Code Usage Ratio for *E. coli* Test Set

shows the usage ratio for each code in the binary coding model set, with penalties for zero. In Figure 6.27 and Figure 6.28 the horizontal axis is the index representation for the codes in the model and the vertical axis is the corresponding usage ratio. As in the equal-weight binary model, there are no notable differences between the usage ratio of the non-penalty and the zero penalty tests. The highest usage ratio value for the *E. coli* test set is ~ 15 and there are more code models associated with higher usage ratios than in the equal-weight binary model. The usage ratios indicate a definite model preference for leader and non-leader test sequences.

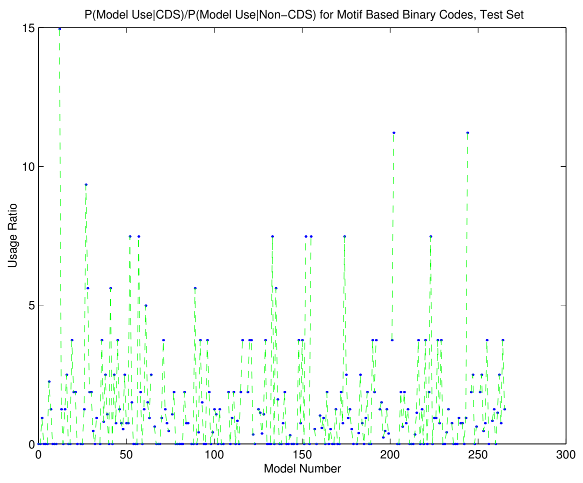


Figure 6.28: Binary Motif-Based Code Usage Ratio for *E. coli* Test Set - Penalty for Zero

6.2.2 Application to Other Prokaryotic Organisms: *S. typhimurium* LT2, *B. subtilis*, and *S. aureus* Mu50

Testing of the *E. coli* test sets suggest that zero-penalty testing produced notably different average syndrome distance and fitness distribution results. Therefore, zero-penalty testing was used to test the binary horizontal code models' performance when applied to *S. typhimurium*, *B. subtilis*, and *S. aureus* test sets. As in the base five code models, the prokaryotic test sets were expected to behave similar to the binary code models, especially in the non-random and Shine-Dalgarno domains. Since *S. typhimurium* is a closer relative of *E. coli*, its syndrome distance values were expected to be lower than those of *B. subtilis* and *S. aureus*.

Equal-Weight Horizontal Codes

Figure 6.29 shows the average syndrome distance value for the equal-weight binary codes tested against the prokaryotic test set. In Figure 6.29 the horizontal axis is position relative to the first base in the initiation codon and the vertical axis is the average syndrome distance

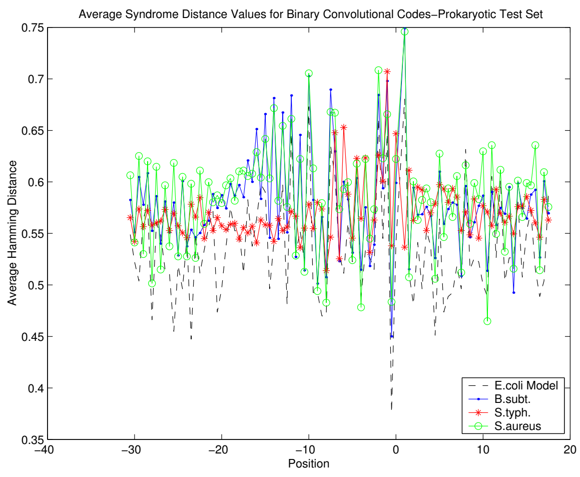


Figure 6.29: Average Syndrome Distance for Prokaryotic Test Set Tested with Binary Code Models

value. As in the *E. coli* test sets, the average syndrome distance of the prokaryotic test set is similar to the model, although in many downstream positions the model has lower syndrome distance values. There are highly periodic regions in the *S. aureus* test set; the first region is from -30 to -20 and the second from -16 to -10. Figure 6.30 shows the average syndrome distance value for the equal-weight, binary codes tested against the prokaryotic test set for position -20 to -9. In Figure 6.30 the horizontal axis is position relative to the first base in the initiation codon and the vertical axis is the average syndrome distance value. In the -20 to -9 region *B. subtilis* and *S. aureus* both have the same periodic behavior. But *S. typhimurium* does not exhibit the same periodicity and has a lower average syndrome distance than *B. subtilis* and *S. aureus*. The -20 to -9 region the closely related prokaryotic organisms syndrome distance are more similar than the syndrome distance values of the distantly related prokaryote. Although interesting, when compared to the *E. coli* leader test set, the *E. coli*'s syndrome distance is more like its distant relatives, *B. subtilis* and *S. aureus* than *S. typhimurium* which shares a common taxonomical lineage with *E. coli*.

Figure 6.31 shows the fitness distribution for the equal-weight binary codes tested against

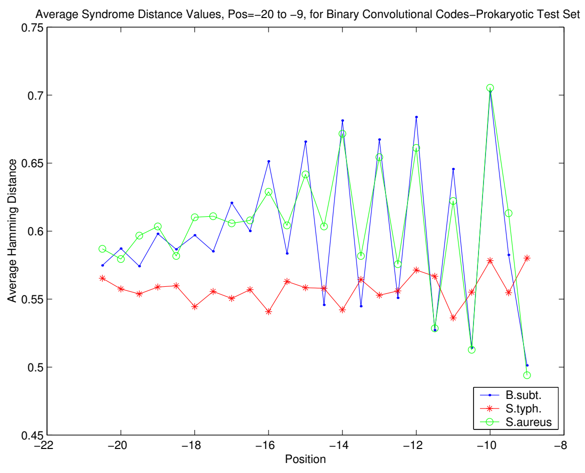


Figure 6.30: Average Syndrome Distance (Position -20 to -9) for Prokaryotic Test Set Tested with Binary Code Models

prokaryotic test set and the fitness distribution for the model. In Figure 6.31 the horizontal

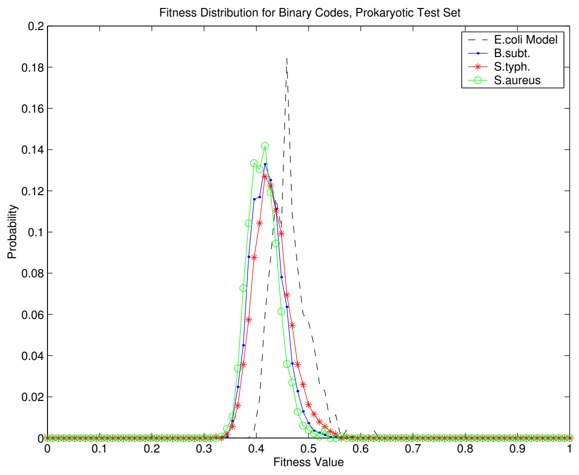


Figure 6.31: Fitness Distribution for Prokaryotic Test Set and Binary Code Models

axis is the fitness values and the vertical axis is the probability of each value occurring. The fitness distribution of the prokaryotic overlaps the model slightly less than the *E. coli* test sets. There is little difference in the fitness distribution of the three prokaryotic test sets.

As in the base-five code model testing analysis, the usage ratio for the prokaryotic test organisms compares the number of times a code is selected as the most fit by an *E. coli* initiation sequence to the number of times the code is selected by the other prokaryotic initiation sequences. Figure 6.32 shows the usage ratio for the *B. subtilis.* Figure 6.33 shows

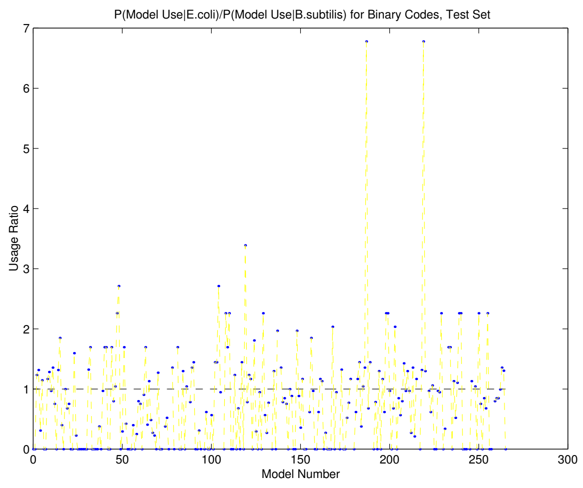


Figure 6.32: Binary Code Usage Ratio for *B. subtilis* Test Set

the usage ratio for the *S. typhimurium* LT2. Figure 6.34 shows the usage ratio for the *S.*

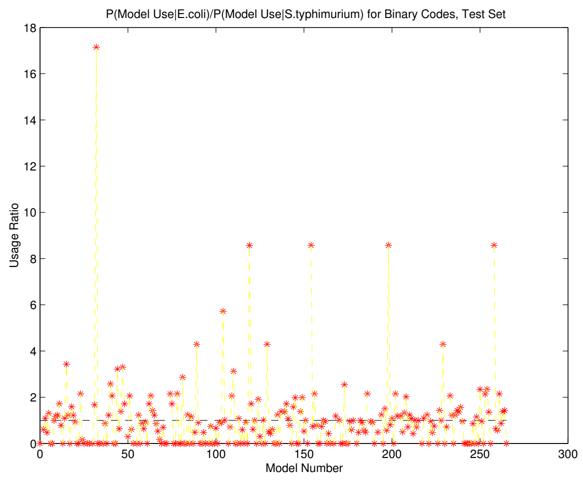


Figure 6.33: Binary Code Usage Ratio for *S. typhimurium* Test Set

aureus Mu50. In Figure 6.32, Figure 6.33, and Figure 6.34 the horizontal axis is the index representation for the codes in the model and the vertical axis is the corresponding usage

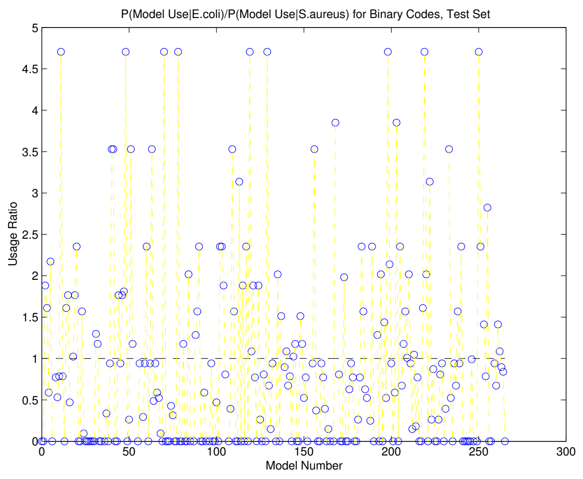


Figure 6.34: Binary Code Usage Ratio for *S. aureus* Test Set

ratio. Unlike the base five model, the highest usage ratio value occurs in *S. typhimurium*, followed by *B. subtilis* and *S. aureus*. The maximum usage ratios for *B. subtilis* and *S. aureus* are lower than the maximum usage ratio for the non-leader test set. The usage ratio clearly show model bias among the prokaryotic test organisms.

Motif-Based Horizontal Codes

Figure 6.35 shows the average syndrome distance value for the motif-based, base five codes tested against the prokaryotic test set. In Figure 6.35 the horizontal axis is position relative to the first base in the initiation codon and the vertical axis is the average syndrome distance value. All three test sets are comparable to the model and they all have global minima distance values at position -14. The average syndrome distance of *S. typhimurium* is slightly higher at -14 than that of *B. subtilis* and *S. aureus* but in regions upstream of -14, *S. typhimurium* has lower average syndrome distance values.

Figure 6.36 shows the fitness distribution for the equal-weight motif-based base five codes tested against prokaryotic test set and the fitness distribution for the model. In Figure 6.36 the horizontal axis is the fitness values and the vertical axis is the probability

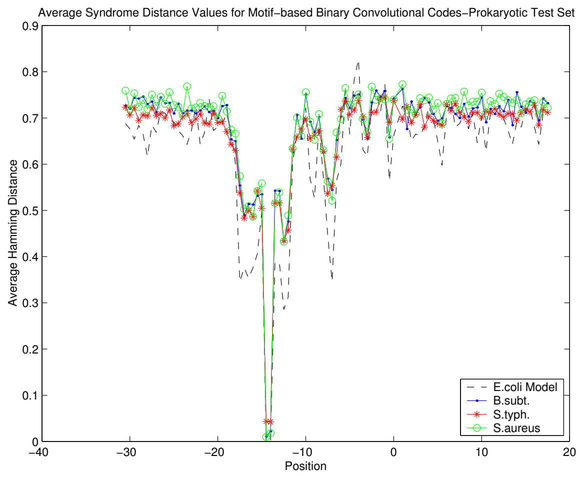


Figure 6.35: Average Syndrome Distance for Prokaryotic Test Set Tested with Motif-Based Binary Code Models

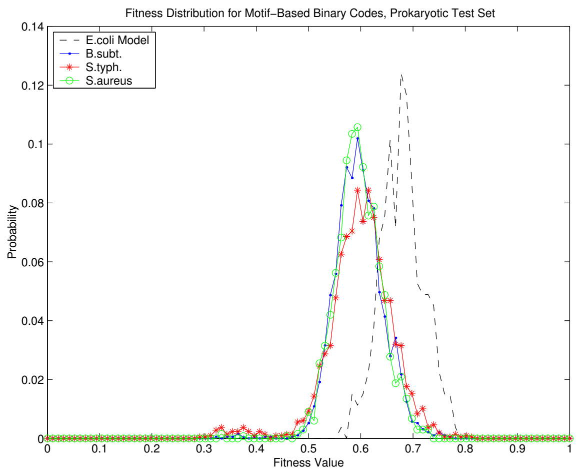


Figure 6.36: Fitness Distribution for Prokaryotic Test Set and Motif-Based Binary Code Models

of each value occurring. The fitness distribution of the prokaryotic test set resembles that of the *E. coli* test set. The fitness distribution for *S. typhimurium* is over a slightly wider region, indicating that *S. typhimurium* may contain more test sequences with relatively high fitness values than *B. subtilis* and *S. aureus*.

Figure 6.37 shows the usage ratio for the *B. subtilis*. Figure 6.38 shows the usage ratio

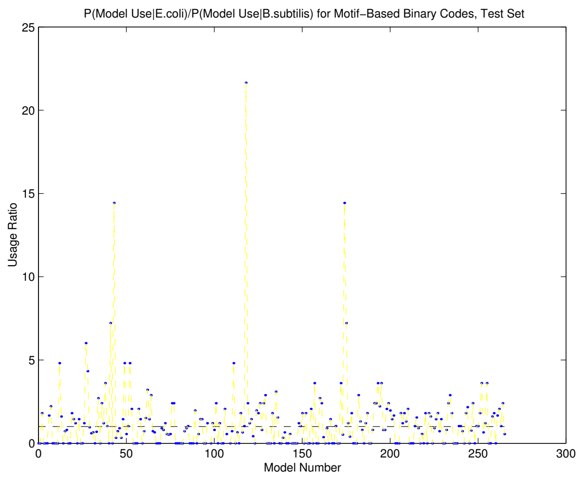


Figure 6.37: Motif-Based Binary Code Usage Ratio for *B. subtilis* Test Set

for the *S. typhimurium* LT2. Figure 6.39 shows the usage ratio for the *S. aureus* Mu50. In

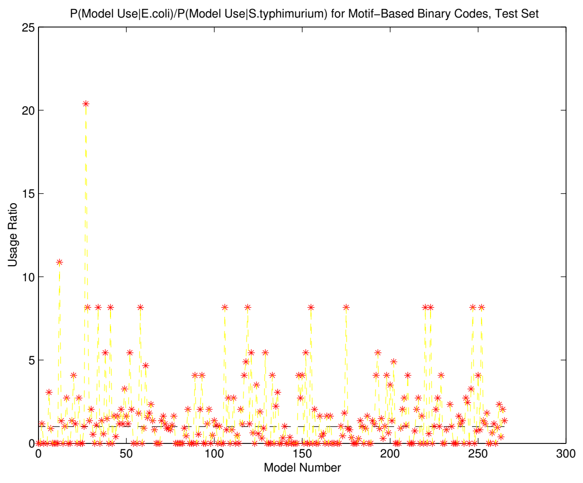


Figure 6.38: Motif-Based Binary Code Usage Ratio for *S. typhimurium* Test Set

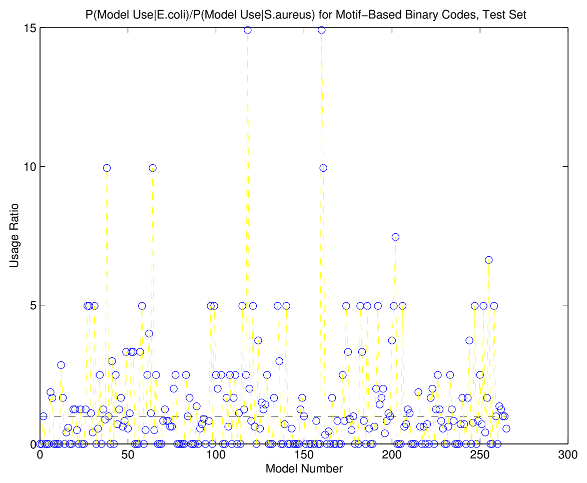


Figure 6.39: Motif-Based Binary Code Usage Ratio for *S. aureus* Test Set

Figure 6.37, Figure 6.38, and Figure 6.39 the horizontal axis is the index representation for the codes in the model and the vertical axis is the corresponding usage ratio. The largest usage ratio occurs in *B. subtilis* followed by *S. typhimurium* and *S. aureus*. Although the maximum usage ratio for *B. subtilis* was higher than the maximum usage ratio for the *E. coli* test set, *B. subtilis* and *S. aureus* have fewer codes with usage ratios above five relative to *E. coli*. Unlike the *E. coli* test set, the prokaryotic test sets' usage ratios for motif-based binary codes are higher than the usage ratios for the equal-weight, binary codes.

6.2.3 Discussion

The *E. coli* test sets fit the binary code models better (particularly with fitness distribution analysis) than the base five code models. Although the distributions of leader and non-leader regions are still similar, their average syndrome distances differ more for the binary code models, particularly the equal-weight, binary model, than in the base five code models. The functional binary code model has the potential of being the better error or non-leader sequence detector when compared to the base five model.

The difference between the non-penalty and zero penalty analyses of the *E. coli* test sets indicates that the non-leader set has a greater number of low-affinity (high zero) binary binding patterns. Using zero penalty fitness analysis with the GA may improve the binary code model. There also needs to be a fitness measure penalizing codes that produce gmasks with large numbers of zero coefficients.

The prokaryotic test sets' results are more similar for the binary code models than for the base five models. There were more notable differences between the prokaryotic test sets in the binary models than the base five models. Although the behavior of the *B. subtilis* and *S. aureus* test sets were more similar to each other than to S. tyhpimurium, the behavior of *S. typhimurium* was not always consistent with the behavior of the *E. coli* test set, to which it is taxonomically more closely related.

Chapter 7

Summary and Conclusion

This work has provided an in-depth analysis of the translation initiation system using table-based, convolutional encoding and syndrome based decoding methods. This work presents and develops a genetic algorithms-based method for the design of optimal convolutional coding models for individual translation initiation sites using *E. coli* K-12 as the model organism. Two types of convolutional coding models were developed and analyzed: sequence based (base five) models and function based (binary) models. The models were applied to *E. coli* leader and non-leader test sequences and to the test leader sequences of three prokaryotic organisms: *S. typhimurium* LT2, *B. subtilis*, and *S. aureus* Mu50. The models were evaluated and the results of applying their associated decoder (represented by the gmask set of each model) were analyzed. The research issues investigated in the analysis include: how well the optimal codes relate to the biology of the system (such as recognition of key initiation-related regions in the mRNA leader), the ability of the code model system to detect errors or non-leader sequences, and the model's ability to recognize other closely and distantly related prokaryotic leaders.

7.1 Analysis of Table-Based Convolutional Coding Models

In the simplest model, there would exist one code that generates every possible mRNA leader, but such a simple model may not be the case. Recall there are other macromolecules besides the ribosome that interact with the mRNA sequence [56][9]; each macromolecule may represent a different decoder having a different code. Additionally, different regions of the 16S rRNA can, as a result of secondary structure, interact with various regions on the mRNA; this suggests that several gmasks derived from more than one code may be required. Therefore, there may be several decoders that decode the mRNA leader. Using the GA based method for convolutional code construction, this work developed two sets of optimal table-based convolutional code models for an *E. coli* training set. The sequence code model used base five, table-based coding principles and the functional code model used binary, table-based coding principles [58].

7.1.1 Code Model Versus Biological Models [8]

It is known that leader regions and other prokaryotic regulatory regions contain similar consensus signals such as the Shine-Dalgarno sequence, the TATA Box and the Pribnow Box [9]. In translation initiation the mRNA anneals to the 3' end of the 16S rRNA in the Shine-Dalgarno (SD) domain [8]. The spacing between the SD domain and the initiation codon affects initiation; the average spacing for initiation is seven base pairs. The average length of complementary sequences between the 5' untranslated region of the mRNA and the last thirteen bases of the 16S rRNA is five bases. The -20 to -13 region, called the non-random domain, also affects translation initiation. This region is thought to assist in correct alignment of the mRNA with the 3' end of the 16S rRNA. In addition to the initiation codon, the second codon is also thought to affect initiation.

In previous work, presented in Chapter 2, the block code model captured several of the above aspects of the translation initiation system, including: the -20 to -13 region, the SD domain, and the initiation codon. The previous convolutional code model, summarized in Chapter 2, captured differences in leader and non-leader sequences, but did not capture specific behavior. Also the convolutional model was not based on a defined encoding algorithm, limiting the ability to develop correction and design heuristics for translation initiation sites.

The current motif-based, base five code model recognizes the non-random domain. Although its next-best, syndrome distance values are in the SD domain, the syndrome distance values are much higher for the SD domain than the non-random domain. The motif-based , binary model recognizes the non-random domain but not as well as the base five model. Some of the decoders produced by the motif-based, base five codes exhibited relatively high similarity to the 3' end of the 16S rRNA. The other base-five and binary code models, including the vertical codes, did not fit the biological initiation model as well as the horizontal motif-based models. The binary, vertical model did capture the region between -10 and the initiation codon better than the other models. All of the binary code models recognized the initiation region. The discovery of optimal code sets that fit several parts of the biological model for initiation is encouraging. Improvements in model development and refinement of the model code set should lead to a better coding based system for describing translation initiation.

7.1.2 Analysis of Models

The sequence and functional models were tested against *E. coli* leader and non-leader sequences and leader sequences of other prokaryotic organisms to analyze: the models' error detecting ability and the models' response to other prokaryotic leader regions. Prokaryotic leader regions have similar Shine-Dalgarno domains. Therefore, a model developed using *E. coli* might perform similarly on other prokaryotic organisms as it does on *E. coli* sequences.

Some of the code models' response to the test sets were consistent with the expectations of this work. The equal-weight, base five models was able to detect errors or non-leader sequences more readily than the other models. In general, the equal-weight models were the better error detectors. The equal-weight, function-based model was able to distinguish the prokaryotic test sequences based on taxonomical relation among the three prokaryotic test sets.

The motif-based code models produced zeros for almost every test sequence, including non-leader sequences, at the position with the greatest weight or error protection, position -15 for base five and -14 for binary. This was unexpected for the non-leader sequence set since the highest weight position is also the position with the greatest binding difference between *E. coli* leader and non-leader sequences. The similar behavior of the non-leader set could be explained by considering the process used to produce the code models. It is highly probable that the GA search space, especially for the motif-based search, contained several local optima based on the fitness criteria. The 266 code set most likely contained several codes that were marginally fit for most of the training set but were one of several optimal codes found for the specific leader being modeled. Such codes may be selected as optimal by only a few leader sequences but have several non-leader or error sequences select it as optimal. This is depicted in the usage ratio analysis which showed that the codes selected as the most fit by the *E. coli* leader set were different from those selected by the non-leader set. As a result, the current code model sets have poor error detection capabilities. Within the code model set, there probably exists a subset of codes with better error detection capabilities than the entire set. The usage ratio analysis, which shows clear code preferences between *E. coli* leader and non-leader sequences, indicates the existence of such a set is plausible. To produce codes with better error-detection capabilities, the code development process should be an iterative process which takes all high fit results and refines them based on the average fitness value when compared against the rest of the training set.

When tested, all the data sets behaved closer to the binary code model than the base-five code model. Differences between near and distant prokaryotic test sets were more evident in the binary models than the base five models. Usage ratio analysis of improved code models may help explain, from a coding theory perspective, the cause of inefficient initiation and regulatory sites for transgenic systems. Also, improving prokaryotic data quality and the code model sets will result in better coding-based taxonomical studies.

The functional binary code model is simpler and has a smaller search space, it also parallels the key element in initiation - hydrogen bond formations. Although the test sequences behave closer to the binary model, the motif-based, base five model captures the biological behavior of the random and SD domains better than the binary model. The results of the binary binding models can be used to improve the base five model because functional model design and sequence model design are complementary processes. Parallel development of both can further the understanding of the link between sequence, structure, and function. This approach is not limited to the study of translation initiation sites, but can be applied to other regulatory sites and protein sequence, structure, and function modeling.

7.2 Research Implications and Contributions

Using table-based coding and genetic algorithms-based code construction techniques, this work was able to construct optimal sequence-based and function-based codes which agree with the biological model. Not all of the code models' performance on the test sets were consistent with the biological model of translation initiation. But, the code models' performance on the training set, especially motif-based code models, was consistent with the biological model of prokaryotic translation initiation. This work also demonstrated that motif-based or unequal error protection codes parallel the biological model better than equal error protection, horizontal and equal error protection ,vertical codes. This finding lends credibility to the nested coding model for translation and other genetic processes.

The results of this work and results from previous work (see Section 2.2) show that we can construct $(3, 1, 4)$ table-based, convolutional code models that: 1) Have decoding gmasks with high similarity to the 3' end of the 16S rRNA; 2) Have decoding masks which identify the non-random and Shine-Dalgarno domains, key regions on the mRNA leader; 3) Have decoding masks that can potentially be used to detect valid and invalid leader sequences. Although the final, and most important, claim has been realized only for the sequence-based, horizontal code model, the hypothesis still holds. The results of this work show that mRNA can be modeled as a noisy, encoded signal and the ribosome as a table-based decoder. The results also indicate that it is feasible to use error-control coding theory to analyze the translation initiation mechanism. As previously stated, the $(3, 1, 4)$ convolutional code is used to demonstrate the GA-based code construction method; longer codes and different rate codes should produce codes that capture the translation initiation system more adequately.

The research presented has contributed to the fields of information theory, coding theory, and the field of computational bioinformatics and biology through the application of information theory, communication theory and coding theory principles to the study and analysis of prokaryotic translation initiation. This work contributed as follows:

- Development and verification of a coding theory view of the translation initiation process in prokaryotic organisms.

- Improvement and discovery of evidence for the nested encoding and decoding model of the genetic communication system.
- Design and implementation of table-based, convolutional coding, model construction techniques for prokaryotic translation initiation systems using syndrome-based methods.
- Construction and analysis of sequence-based, error-control coding models for prokaryotic translation initiation.
- Construction and analysis of function-based, error-control coding models for prokaryotic translation initiation.
- Comparative testing and analysis of coding models for prokaryotic organisms of differing taxonomical relatedness to the model organism.
- Exploration of the use of coding based models to investigate taxonomical relatedness based on regulatory sites.
- Extension of table-based coding principles to field five convolutional codes.

7.3 Future Research

The strength of the genetic algorithm-based model design technique proposed in this work is that the model can be improved by including additional parameters and restraints based on biological model and data analysis information (from databases, micro array expression profiles, etc.) to the GA's fitness evaluation routine. Such information may improve the model produced. The method presented and implemented in this work allows the design of convolutional coding models based on sequence and function. Both models complement each other and can be used to develop an optimal coding-based representation of the genetic system. In the immediate future, research will focus on:

- Inclusion of usage ratio information to refine current coding model.

- Development of a discriminating GA that uses a set of intergenic non-leader or randomly generated subsequences to construct convolutional code models with better error detection capabilities. The discriminating GA will construct the best code in a quasi-species sense; codes with near optimal performance that classify leader regions as correctable sequences and recognize non-leaders as error sequences outside of its error correction threshold.
- Inclusion of "correctibility" fitness measure. Eigen's quasi-species theory suggests that perfect sequences do not exist if an organism is to be evolutionarily viable [12]. Therefore the existing leader sequence set must not be the correct (or errorless) leader sequence set. They must, like the evolutionarily viable species, contain errors near an error threshold. Assuming the leader sequence used to construct the model contains errors, the best model would be the one that is near optimal when applied to the leader sequence but becomes optimal when syndrome based correction is applied to the leader sequence.
- Construction of a combination coding system with motif-based codes that recognize different regions of the leader sequence. Decoding decisions will be based on the combined output of the separate motif-based decoders.
- Exploration of convolutional coding models with longer memories and different coding rates.
- Construction of sequence model codes which permit zeros in the generators.
- Extension of the binary-binding, functional code model to tertiary codes; using three symbols to represent not only binding and non-binding, but also the number of hydrogen bonds formed when binding occurs.
- Exploration of non-linear coding models, particularly for binary coding models.
- Multi-dimensional code models for translation initiation.

The success of this work will contribute to the functional understanding of genetic regulatory regions, development of efficient initiation and regulation sites for transgenic systems, and contribute to the development and realization of *in silico* tools for simulating genetic processes. Current research results are encouraging. Research into improved coding theory models for translation initiation and related genetic processes continues.

List of References

[1] G. Battail. Does information theory explain biological evolution? . *Europhysics Letters*, 40(3):343–348, November 1997.

[2] Thomas D. Schneider. Theory of Molecular Machines. I. Channel Capacity of Molecular Machines. *Journal of Theoretical Biology*, 148:83–123, 1991.

[3] Lila Kari, Rob Kitto, and Gabriel Thierrin. Codes, Involutions and DNA Encodings. University of Western Ontario, London, Ontario, Canada. Submitted.

[4] Elebeoba E. May, Mladen A. Vouk, Donald L. Bitzer, and David I. Rosnick. Coding Model for Translation in E. coli K-12 . In *First Joint Conference of EMBS-BMES.*, 1999.

[5] Elebeoba E. May, Mladen A. Vouk, Donald L. Bitzer, and David I. Rosnick. The Ribosome as a Table-Driven Convolutional Decoder for the Escherichia coli K-12 Translation Initiation System . In *World Congress on Medical Physics and Biomedical Engineering Conference.*, 2000.

[6] Elebeoba E. May, Mladen A. Vouk, Donald L. Bitzer, and David I. Rosnick. Coding Theory Based Maximum-Likelihood Classification of Translation Initiation Regions in Escherichia coli K-12 . In *2000 Biomedical Engineering Society Annual Meeting.*, 2000.

[7] David A. Coley. *An Introduction to Genetic Algorithms for Scientists and Engineers.* World Scientific Publishing Co. Pte. Ltd., Singapore, 1999.

[8] Larry Gold and Gary Stormo. Translational Initiation. In *Escherichia coli and Salmonella typhimurium, Cellular and Molecular Biology*, pages 1302–1307, 1987.

[9] Benjamin Lewin. *Genes V.* Oxford University Press, New York, NY, 1995.

[10] Thomas D. Schneider. Information content of individual genetic sequences . *Journal of Theoretical Biology*, 189:427–441, 1997.

[11] Elebeoba E. May. Comparative Analysis of Information Based Models for Initiating Protein Translation in Escherichia coli K-12. Master's thesis, NCSU, December 1998.

[12] Manfred Eigen. The origin of genetic information: viruses as models . *Gene*, 135:37–47, 1993.

[13] James W. Fickett. The gene identification problem: an overview for developers. *Computers and Chemistry*, 20(1):103–118, 1996.

[14] John Henderson, Steven Salzberg, and Kenneth H. Fasman. Finding Genes in DNA with a Hidden Markov Model. *Journal of Computational Biology*, pages 127–1441, 1997.

[15] Alexander V. Lukashin and Mark Borodovsky. GeneMark.hmm: New Solutions for Gene Finding. *Nucleic Acids Research*, 26(4):1107–1115, 1998.

[16] Rajasekhar Raman and G. Christian Overton. Application of hidden markov modeling in the characterization of transcription factor binding sites. In *Proceedings of the Twenty-Seventh Annual Hawaii International Conference on System Sciences*, 1994.

[17] A. Krogh, I. Mian, and D. Haussler. A Hidden Markov Model that Finds Genes in E. Coli DNA . *Nucleic Acids Research*, 22:4768–4778, December 1994.

[18] William N. Grundy, Timothy L. Bailey, Charles P. Elkan, and Michael E. Baker. Meta-MEME: Motif-based hidden Markov models of protein families . *Computers Applications in the Biosciences*, 13(4):397–406, 1997.

[19] Sean R. Eddy. Hidden Markov Models . *Current Opinion In Structural Biology*, 6:361–365, 1996.

[20] Ajay Dholakia. *Introduction to Convolutional Codes with Applications*. Kluwer Academic Publishers, Norwell, Massachusetts, 1994.

[21] Mark Borodovsky and James McIninch. GENEMARK: Parallel Gene Recognition for Both DNA Strands . *Computers and Chemistry*, 17(2):123–133, 1993.

[22] Irena Cosic. *The Resonant Recognition Model of Macromolecular Bioactivity: Theory and Applications*. Birkhauser Verlag, Basel, Switzerland, 1997.

[23] A. Arneodo, Y. d'Aubenton Carafa, E. Bacry, P. V. Graves, J. F. Muzy, and C. Thermes. Wavelet based fractal analysis of DNA sequences. *Physica D*, pages 1–30, 1996.

[24] V. Veljkovic, I. Cosic, B. Dimitrijevic, and D. Lalovic. Is it Possible to Analyze DNA and Protein Sequences by the Methods of Digital Signal Processing. *IEEE Transactions on Biomedical Engineering*, BME-32(5):337–341, May 1985.

[25] Edward C. Uberbacher and Richard J. Mural. Locating Protein-Coding Regions in Human DNA Sequences by a Multiple Sensor-Neural Network Approach . *Proceedings of the National Acadamy of Science, USA*, 88:11261–11265, December 1991.

[26] William R. Pearson. Protein sequence comparison and protein evolution. In *Intelligent Systems for Molecular Biology*, 1998.

[27] Elizabeth Pennisi. Ideas fly at gene finding jamboree . *Science*, 287(5461):2182–2184, March 2000.

[28] Mark D. Adams, Susan E. Celniker, Robert A. Holt, and et al. The genome sequence of Drosophila melanogaster . *Science*, 287(5461):2185–2195, March 2000.

[29] Ramon Roman-Roldan, Pedro Bernaola-Galvan, and Jose L. Oliver. Application of information theory to DNA sequence analysis: a review. *Pattern Recognition*, 29(7):1187–1194, 1996.

[30] Rina Sarkar, A. B. Roy, and P. K. Sarkar. Topological Information Content of Genetic Molecules – I . *Mathematical Biosciences*, 39:299–312, 1978.

[31] T. B. Fowler. Computation as a thermodynamic process applied to biological systems . *International Journal of Bio-Medical Computing*, 10(6):477–489, 1979.

[32] K. Palaniappan and M. E. Jernigan. Pattern analysis of biological sequences . In *Proceedings of the 1984 IEEE International Conference on Systems, Man, and Cybernetics*, 1984.

[33] Hagai Almagor. Nucleotide distribution and the recognition of coding regions in DNA sequences: an information theory approach. *Journal of Theoretical Biology*, 117:127–136, 1985.

[34] Thomas D. Schneider. Theory of Molecular Machines. II. Energy Dissipation from Molecular Machines. *Journal of Theoretical Biology*, 148:125–137, 1991.

[35] Stephen F. Altschul. Amino Acid substitution matrices from an information theoretic perspective . *Journal of Molecular Biology*, 219:555–565, 1991.

[36] Peter Salamon and Andrezej K. Konopka. A maximum entropy principle for the distribution of local complexity in naturally occuring nucleotide sequences . *Computers and Chemistry*, 16(2):117–124, 1992.

[37] J. L. Oliver, P. Bernaola-Galvan, J. Guerrero-Garcia, and R. Roman-Roldan. Entropic profiles of DNA sequences through chaos-game-derived images . *Journal of Theoretical Biology*, 160:457–470, 1993.

[38] Francisco M. De La vega, Carlos Cerpa, and Gariel Guarneros. A mutual information analysis of tRNA sequence and modification patterns distinctive of species and phylogenetic domain. In *Pacific Symposium on Biocomputing*, pages 710–711, 1996.

[39] Thomas D. Schneider and David N. Mastronarde. Fast multiple alignment of ungapped DNA sequences using information theory and a relaxation method . *Discrete Applied Mathematics*, 71:259–268, 1996.

[40] Bonnie J. Strait and T. Gregory Dewey. The Shannon information entropy of protein sequences . *Biophysical Journal*, 71:148–155, 1996.

[41] Angelo Pavesi, Bettina De Iaco, Maria Ilde Granero, and Alfredo Porati. On the informational content of overlapping genes in prokaryotic and eukaryotic viruses . *Journal of Molecular Evolution*, 44(6):625–631, 1997.

[42] David Loewenstern and Peter N. Yianilos. Significantly lower entropy estimates for natural DNA sequences. In *Proceedings of the Data Compression Conference*, 1997.

[43] Thomas D. Schneider. Measuring molecular information . *Journal of Theoretical Biology*, 201:87–92, 1999.

[44] Thomas D. Schneider, Gary D. Stormo, Larry Gold, and Andzej Dhrenfeucht. Information Content of Binding Sites on Nucelotide Sequences. *Journal of Molecular Biology*, 188:415–431, 1986.

[45] Thomas D. Schneider and R. Michael Stephens. Sequence Logos: a New Way to Display Consensus Sequences . *Nucleic Acids Research*, 18(20):6097–6100, September 1990.

[46] Richard E. Blahut. *Theory and Practice of Error Control Codes.* Addison-Wesley Publishing Company, Inc., Reading, MA, 1983.

[47] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory* . John Wiley and Sons, Inc., New York, N.Y., 1991.

[48] Didier G. Arques and Christian J. Michel. A code in the protein coding genes. *BioSystems*, 44:107–134, 1997.

[49] Nikola Stambuk. On circular coding properties of gene and protein sequences. *Croatica Chemica ACTA*, 72(4):999–1008, 1999.

[50] Nikola Stambuk. Symbolic Cantor Algorithm (SCA): A method for analysis of gene and protein coding . *Periodicum Biologorum*, 101(4):355–361, 1999.

[51] Nikola Stambuk. On the genetic origin of complementary protein coding . *Croatica Chemica ACTA*, 71(3):573–589, 1998.

[52] Elwyn R. Berlekamp. *Algebraic Coding Theory.* McGraw-Hill Book Company, New York, NY, 1968.

[53] A. Arneodo, Y. d'Aubenton Carafa, E. Bacry, P. V. Graves, J. F. Muzy, and C. Thermes. Wavelet based fractal analysis of DNA sequences. *Physica D*, pages 1–30, 1996.

[54] Peter Sweeney. *Error Control Coding an Introduction.* Prentice Hall, New York, NY, 1991.

[55] Shu Lin and Daniel J. Costello Jr. *Error Control Coding: Fundamentals and Applications.* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1983.

[56] J. Watson, N. Hopkins, J. Roberts, J. Steitz, and A. Weiner. *Molecular Biology of the Gene.* The Benjamin Cummings Publishing Company, Inc., Menlo Park, CA, 1987.

[57] Elebeoba E. May, Mladen A. Vouk, Donald L. Bitzer, and David I. Rosnick. Analysis of Coding Theory Based Models for Initiating Protein Translation in Prokaryotic Organisms. *IEEE Transactions on Information Technology in BioMedicine*, Submitted, May 2001.

[58] Donald L. Bitzer and Mladen A. Vouk. A Table-Driven (Feedback) Decoder. In *Tenth Annual International Phoenix Conference on Computers and Communications*, pages 385–392, 1991.

[59] Donald L. Bitzer, Mladen A. Vouk, and Ajay Dholakia. Genetic Coding Considered as a Convolutional Code. North Carolina State University, Raleigh, 1992.

[60] Robert Kotrys and Piotr Remlein. The Genetic Algorithm used in search of the good TCM codes. In *4th Inetrnational Workshop on Systems, Signals and Image Processing, IWSSIP'97.*, pages 53–57, 1997.

[61] Richard D. Wesel. Turbo code design for high spectral efficiency. University of California, Los Angeles, CA, 2000.

[62] Tadashi Wadayama, Koichiro Wakasugi, and Masao Kasahara. An 8-Dimensional Trellis-Coded 8-PSK with Non-Zero Crossing Constraint. *IEICE Trans. Fundamentals*, E77-A(8):1274–1280, August 1994.

[63] Tiffany M. Barnes. Using Genetic Algorithms to Find the Best Generators for Half-Rate Convolutional Coding. North Carolina State University, Raleigh, NC, 1994.

[64] Yu. G. Savchenko and A. A. Svistel'nik. An Approach to Pattern Recognition Systems. *Engineering Cybernetics*, (2):144–146, 1968.

[65] David I. Rosnick. *Free Energy Periodicity and Memory Model for E. coli Codings.* PhD thesis, North Carolina State University, Raleigh, NC, 2001.