2009

Investigation and quantification of codon usage bias trends in prokaryotes

Amanda L. Hanes
Wright State University

Investigation and quantification of codon usage bias trends in prokaryotes

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

By

Amanda L. Hanes
B.S.C.S., Wright State University, 2006

2009
Wright State University

WRIGHT STATE UNIVERSITY
SCHOOL OF GRADUATE STUDIES

June 5, 2009

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY AMANDA L. HANES ENTITLED INVESTIGATION AND QUANTIFICATION OF CODON USAGE BIAS TRENDS IN PROKARYOTES BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE.

Michael L. Raymer, Ph.D.
Thesis Director

Thomas Sudkamp, Ph.D.
Department Chair

Committee on
Final Examination

Michael L. Raymer, Ph.D.

Travis E. Doom, Ph.D.

Dan E. Krane, Ph.D.

Joseph F. Thomas, Jr., Ph.D.
Dean, School of Graduate Studies

ABSTRACT

Hanes, Amanda L. M.S., Department of Computer Science and Engineering, Wright State University, 2009. Investigation and quantification of codon usage bias trends in prokaryotes.

Organisms construct proteins out of individual amino acids using instructions encoded in the nucleotide sequence of a DNA molecule. The genetic code associates combinations of three nucleotides, called codons, with every amino acid. Most amino acids are associated with multiple synonymous codons, but although they result in the same amino acid and thus have no effect on the final protein, synonymous codons are not present in equal amounts in the genomes of most organisms. This phenomenon is known as codon usage bias, and the literature has shown that all organisms display a unique pattern of codon usage. Research also suggests that organisms with similar codon usage share biological similarities as well. This thesis helps to verify this theory by using an existing computational algorithm along with multivariate analysis to demonstrate that there is a significant difference between the codon usage of free-living prokaryotes and that of obligate intracellular prokaryotes. The observed difference is primarily the result of GC content, with the additional effect of an unknown factor.

Although the existing literature often mentions the strength of biased codon usage, it does not contain a clear, consistent definition of the concept. This thesis provides a disambiguated definition of bias strength and clarifies the relationships between this and other properties of biased codon usage. A bias strength metric, designed to match the given definition of bias strength, is proposed. Evaluation of this metric demonstrates that it compares favorably with existing metrics used in the literature as criteria for bias strength, and also suggests that codon usage bias in general follows the trend of being either strong and global to the genome, or weak and present in only a subset of the genome. Analysis of these metrics provides insight into the unknown factor partially responsible for the codon usage difference between free-living and obligatorily intracellular prokaryotes, and the proposed bias strength metric is used to draw conclusions about the characteristics of GC-content bias.

Table of Contents

List of Figures

List of Tables

1. Introduction

1.1. Overview

The genetic code describes the manner in which the genetic material, DNA, encodes instructions for building and regulating the production of proteins. DNA (deoxyribonucleic acid) molecules are chains (or polymers) of four building blocks called nucleotides. Most of the information encoded in DNA controls the synthesis of proteins, which are themselves polymers of amino acids. There are twenty commonly found amino acids; a typical protein consists of one or more chains of around 300 amino acids. These proteins are encoded in DNA using groups of three nucleotides, called codons, to indicate specific amino acids. Most amino acids are associated with multiple synonymous codons, but although they represent the same amino acid these synonymous codons are not found in equal proportions in DNA. The unequal usage of synonymous codons within an organism's DNA is known as codon usage bias.

Many different factors have been identified as causes of codon usage bias, and the combination of these effects produces a unique codon usage pattern in every organism. Some are associated with making the organism more biologically efficient, others with adapting the organism to a certain environment. Similarities in these patterns have been used to identify some degrees of biological relationship among groups of organisms.

The biological significance of synonymous codon usage trends lies in the fact that this is one of only a few forms of adaptation that takes place at the level of the storage of genetic information rather than at the level of biological functionality. The fact that this variation has no effect whatsoever on the products of an organism's genes implies that evolution operates a finer molecular level than that of amino acids and proteins. Further investigation of this evolutionary mechanism will provide a greater understanding of its effects on different types of organisms, enabling greater insight into the workings of evolution as a whole.

1.2. Current research

Carbone *et al* (Carbone, Kepes et al. 2005) have shown that it is possible to distinguish thermophilic from mesophilic organisms as well as among organisms with several different respiratory characteristics on the basis of codon usage bias. The same work also demonstrated that organisms with different types of bias were separable in the same manner, and suggested that codon usage bias can be thought of as a multi-dimensional feature space where the distance between two organisms is a function of their biological similarity. Heizer, Raiford *et al* showed that there are some exceptions to this trend. The codon usage of some organisms is determined primarily by the biosynthetic cost of amino acids, the effect of which overrides that of lifestyle (Heizer, Raiford et al. 2006).

The existing literature in this area makes mention of several metrics that measure aspects of a genome's codon usage bias in a computational manner. Although their use in the literature is limited, such metrics can provide information about the biology of an organism by applying simple computational techniques to a mathematical representation of a codon usage pattern.

1.3. Contribution

This thesis will extend the study of codon usage bias as a genomic comparison tool by applying existing computational and analytic techniques to previously unexplored types of organisms. If new types of organisms are separable in the same way as previously-studied groups, this will further validate the idea of codon usage space as a means of determining biological similarity among organisms.

The possibility of deriving biological insight from codon usage bias using computational means will also be explored. Issues with existing methods for assessing both the strength of a particular bias, and the degree of adherence of a gene or genome to that bias will be addressed, and a new metric for quantifying bias strength will be proposed and evaluated against existing methods to determine whether this type of biological study is viable.

2. Background & literature review

2.1. The genetic code

In order to fully understand the uses and implications of codon usage bias in the following computations and analyses, it is necessary to first have an understanding of the biological context in which it occurs. The following section provides such an understanding via a discussion of basic molecular biology: DNA and the genome, proteins, and the biological processes and flow of genetic information involved in synthesizing the latter from the former.

2.1.1. The genome

The complete set of an organism's genetic information is called its genome. This information comprises all of the genetic information required by an organism in order to grow, reproduce, and pass on its traits to its offspring. These tasks, or rather the biological functions that comprise them, are accomplished at the molecular level by biological molecules called proteins. Often referred to as the "building blocks of life," proteins are the basic units of biological functionality and structure. Since proteins are responsible for nearly every biological function, it follows that an organism's viability is dictated largely by its ability to produce proteins not only correctly, but also efficiently. Some proteins, for example, are useful only under certain conditions, such as high temperature or when the organism has ingested a particular nutrient. Producing these specialized proteins when they are not needed wastes energy and resources that could be used to produce other, useful proteins, making the organism inefficient and ill-suited to survive. The purpose of the genome is to store instructions for producing all the proteins the organism needs, as well as regulation mechanisms that ensure that each protein is synthesized only when necessary.

2.1.2. DNA

DNA (deoxyribonucleic acid) is the genetic material, the medium in which genetic information is stored. An organism's genome is organized into one or more units called chromosomes, chains of DNA that can form closed loops or long strands. Within each chromosome are regions called genes, each of which contains instructions for synthesizing a gene product (usually a protein) and may be associated with a regulatory region of the DNA strand, which indicates when that gene product (protein) should be synthesized. Also included in the genome are stretches of DNA that do not contain genes or regulation mechanisms. These regions have no known biological function, and are sometimes known as junk DNA. The remainder of this thesis will be primarily concerned with the portions of the genome that contain protein-coding genes (also known as the coding sequences) and will largely ignore the regulatory and junk DNA areas.

The storage mechanism of a DNA molecule is a four-character "alphabet" of nucleotides combined together in a linear chain to form DNA. The four nucleotides are adenine, guanine, cytosine, and thymine (commonly abbreviated A, G, C, and T). Information in a

DNA chain is thus stored as a particular combination of A's, G's, C's, and T's, just as words are formed in the English language by using particular combinations of letters.

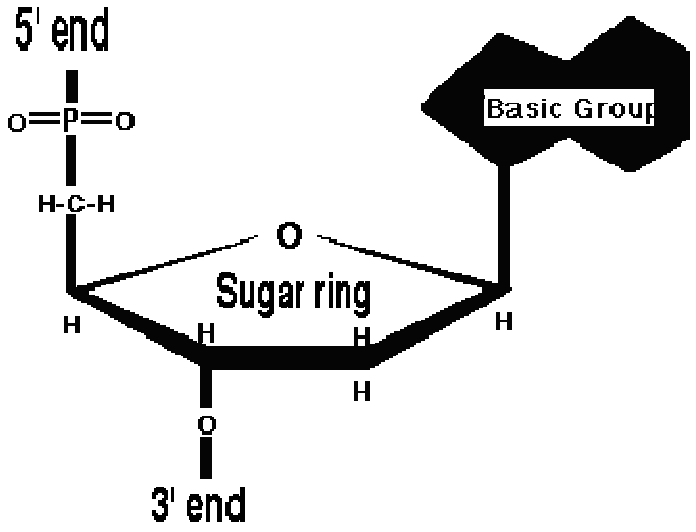

Figure 1. Structure of a nucleotide

The structure of a nucleotide consists of a phosphate group, a deoxyribose sugar, and a nitrogenous base (see Figure 1). While the phosphate and sugar are identical among the four nucleotides, the nitrogenous base identifies the nucleotide as an A, G, C, or T. The chain of nucleotides that forms a DNA molecule is held together by phosphodiester bonds, which form between the phosphate group of one nucleotide and the deoxyribose sugar of the next (Krane and Raymer 2003). This gives the molecule directionality; the end of the strand with the exposed phosphate group is the 5' end and the end with the exposed sugar is the 3' end. The sequence of nucleotides is read from 5' to 3'. A DNA molecule consists of two of these chains in an anti-parallel configuration, where the 5' end of one strand coincides with the 3' end of the other. The molecule is held together by bonds that form between the nitrogenous bases on the two strands. Because of the angle of the phosphodiester bonds, the two strands wrap around each other, giving the DNA molecule its characteristic double helix configuration (see Figure 2).

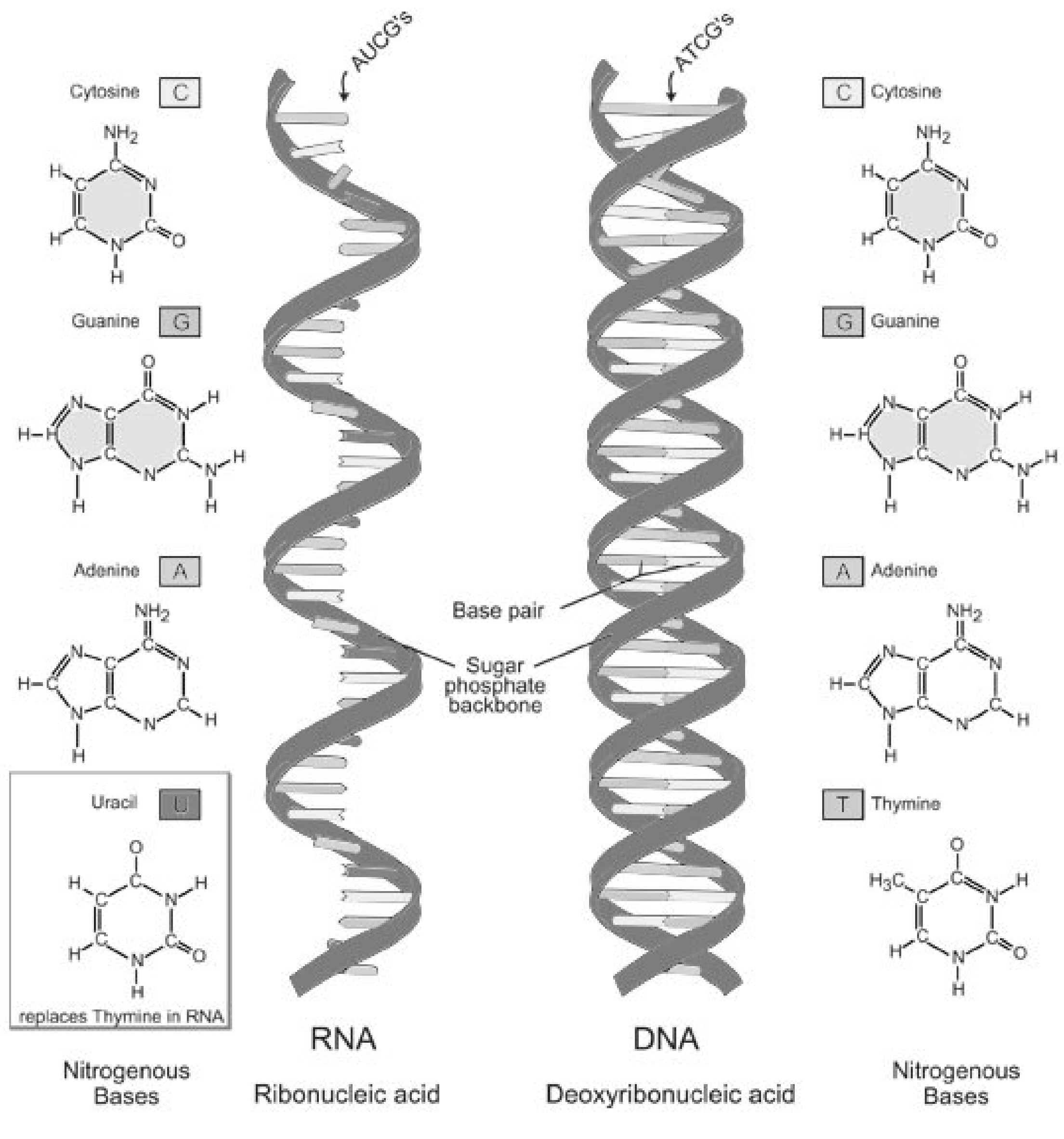

Figure 2. Double-helix configuration of DNA
Adapted from (NHGRI 2009). Image resides at URL:
www.genome.gov/Pages/Hyperion/DIR/VIP/Glossary/Illustration/rna.shtml

The bonds between the nitrogenous bases only form between particular pairs of nucleotides in a process called complementary base pairing. Adenine pairs with thymine and guanine pairs with cytosine. The information on the two parallel strands in a DNA molecule is therefore redundant, as each strand is the reverse complement of the other. That is, one can obtain the sequence of one strand by reading the sequence of the other in reverse (3' to 5') and replacing each nucleotide with its complement (A's with T's, G's with C's, etc.). Genes can be located on either strand; the strand from which a gene is being read is known as the sense strand. This is generally the sequence that is provided when discussing genomic sequences. The two strands of DNA are known as the leading and lagging strand according to their behavior during the process of DNA replication. For the purposes of this work, the actual mechanics of the replication process are irrelevant; it is necessary only to note that the leading strand is the strand on which replication begins.

2.1.3. Proteins

Proteins are chains of amino acids synthesized from the information stored in DNA. After it is synthesized, a protein folds into a unique three-dimensional structure determined by its amino acid sequence. It is well accepted by molecular biologists that protein function is a result of three-dimensional structure, which is itself largely determined by amino acid sequence (cite Anfinsen). The twenty different amino acids can be divided into three different functional groups: hydrophobic, polar, and charged. These groups have specific biological and chemical properties; there is further variation among the amino acids belonging to any particular group. Consequently, each amino acid has unique properties that make it behave differently when included in a protein than any other amino acid. The substitution, addition, or removal of one or more amino acids in a protein can result in changes in the protein's structure, and thus its biological functionality. Because an organism’s fitness is almost entirely dependent on its ability to produce functioning proteins, any change to an amino acid sequence is potentially disastrous.

2.1.4. Central dogma

The biological mechanisms and flow of genetic information involved in the process of synthesizing proteins from DNA are described by a concept commonly known as the central dogma of molecular biology. The central dogma states that genetic information flows from DNA to RNA to proteins. RNA (ribonucleic acid) is a single-stranded chain of nucleotides synthesized from a DNA template by proteins called RNA polymerases. An RNA molecule is a direct copy of its DNA counterpart with regards to its information content; the differences between the two molecules are that in RNA, thymine (T) is replaced by uracil (U), and RNA is a single-stranded molecule. RNA molecules also possess one additional 3’ oxygen molecule relative to DNA. The information in the RNA molecule is then used as a template for the protein’s corresponding sequence of amino acids in a process called translation.

2.1.5. The genetic code

Proteins are composed of twenty different amino acids, while DNA has only four nucleotides. Therefore, in order to translate a sequence of nucleotides into a chain of amino acids, it is necessary to use three nucleotides to indicate one amino acid. Combining four different nucleotides in three-nucleotide groups gives us 64 possible combinations, or codons. Each codon is associated with a single amino acid, with the exception of three termination codons that are used to indicate the end of a gene sequence. Because there are more codons than amino acids, most amino acids are associated with two to four synonymous codons, with the exception of methionine and tryptophan which have one codon each (Table 1).

Table 1. The genetic code

Amino Acid	Codons
Methionine (Met)	ATG
Tryptophan (Trp)	TGG
Lysine (Lys)	AA(A,G)
Asparagine (Asn)	AA(C,T)
Glutamine (Gln)	CA(A,G)
Histidine (His)	CA(C,T)
Glutamic acid (Glu)	GA(A,G)
Aspartic acid (Asp)	GA(C,T)
Tyrosine (Tyr)	TA(C,T)
Cysteine (Cys)	TG(C,T)
Phenylalanine (Phe)	TT(C,T)
Isoleucine (Ile)	AT(A,C,T)
Threonine (Thr)	AC*
Proline (Pro)	CC*
Alanine (Ala)	GC*
Glycine (Gly)	GG*
Valine (Val)	GT*
Arginine (Arg)	CG* | AG(A,G)
Leucine (Leu)	CT* | TT(A,G)
Serine (Ser)	TC* | AG(C,T)
Termination	TA(A,G) | TGA

2.1.6. Translation

Translation is the process by which a protein is synthesized from its RNA template (messenger RNA, or mRNA). The biomolecules involved in this process are ribosomes, which attach new amino acids to the growing protein chain, and transfer RNA (tRNA), relatively small RNA molecules that recruit amino acids to add to the chain. The amino acid to codon match is accomplished by complementary base pairing; each transfer RNA contains an anticodon that complements a codon for its amino acid. After binding an amino acid, the transfer RNA base-pairs with the appropriate codon on the mRNA template, thus positioning it for the ribosome to add to the growing protein and continue to the next codon. There is one specific transfer RNA molecule for every codon-amino acid pair, but some transfer RNAs are isoaccepting. An isoacceptor recognizes similar synonymous codons in addition to its own.

2.1.7. Biased usage of codons

Because there are 64 possible codons and only twenty amino acids, the code contains some degeneracy. One might expect that one synonymous codon is essentially the same as any other, since using one over another does not change which amino acid is included in the protein. If this were the case, synonymous codons should appear in coding sequences with approximately equal frequency. However, research has demonstrated that this is not the case (Grantham, Gautier et al. 1980). Synonymous codons are not used in equal proportion; additionally, the usage of synonymous codons varies sharply in different genomes.

The significance of codon usage bias is that it is evidence of an evolutionary mechanism that has nothing to do with an organism's physical characteristics. One view of evolution emphasizes selective pressure at the protein level; a mutation to a DNA sequence that changes the function of a protein persists and eventually becomes fixed in that species' genome if it improves the fitness of the organism by changing protein composition, and thus structure and function. Codon usage bias constitutes mutations that do not modify the protein composition of the organism. Rather, the choice of particular codons over others may improve an organism's fitness on a level more subtle than that of protein-level phenotype.

2.2. Literature review: codon usage bias

Codon usage bias was first identified in the 1980's. Grantham *et al* found that synonymous codons did not appear in genomes with equal frequency, and noted that the genomes of closely related organisms contained similarly biased codon usage (Grantham 1980), (Grantham, Gautier et al. 1980). Subsequent work by Ikemura demonstrated that all tRNAs are not equally abundant within an organism, and established a correlation between codon usage and tRNA population in several organisms (Ikemura 1981). Others went on to confirm that a positive correlation existed between the degree of biased codon usage in a gene and the gene's level of expression (Gouy and Gautier 1982), (Bennetzen and Hall 1982). This work suggested that the observed correlation was the result of a translational efficiency bias in highly-expressed genes, in which the use of codons corresponding to abundant tRNAs allowed these genes to be translated more efficiently by decreasing the time needed for tRNA recruitment and amino acid incorporation. Bulmer observed that this theory did not account for the presence of codon usage bias in lowly-expressed genes, and postulated that bias could be a result of the combined effects of selection, mutation, and genetic drift (Bulmer 1991). From this point in the literature onward, research in this area has fallen into three broad categories: quantifying codon usage bias, identifying different types of bias, and determining the evolutionary mechanisms responsible for biased usage.

2.2.1. Evolutionary causes of codon usage bias

Since the discovery of biased synonymous codon usage, one of the major outstanding questions has been why some synonymous codons are preferred over others. Early theories assumed that strongly biased usage was a result of an organism selecting codons on the sole basis of translational efficiency. These theories provide an explanation for the presence of bias in highly-expressed genes, but do not account for the biased usage observed in weakly-expressed genes. If selection for translational efficiency were the sole cause of codon usage bias, one would expect to see the effects of the bias primarily in genes that are expressed frequently because there the consequences of inefficiency are compounded. Genes that are expressed less often would not experience as strong a selective pressure towards efficiency, and thus would not display codon usage bias to the degree of highly-expressed genes. Two conflicting theories were brought forth to explain the existence of codon usage bias in weakly-expressed genes: the expression-regulation theory and the selection-mutation-drift theory. The expression-regulation theory stated that rare codons are used in weakly-expressed genes in order to keep their expression low (Hinds and Blake 1985), (Konigsberg and Godson 1983). Although it is the case that weakly-expressed genes contain more non-preferred codons than do highly-expressed genes, a causative relationship was never proven. This theory was quickly supplanted by the selection-mutation-drift theory (Bulmer 1991), which stated that codon usage patterns are a result of a balance between selection favoring the preferred codons and mutational drift allowing the non-preferred codons to persist. The effect of selection on codon usage bias is widely accepted, but the role of mutation has not been conclusively determined. Recent work by Vetsigian and Goldenfeld (Vetsigian and Goldenfeld 2009) proposed a coevolutionary theory in which both mutation and selection pressures influence the codon usage in a genome, which in turn affects cellular resources such as nucleotide and tRNA availability. Optimizing the allocation of these resources affects the mutation and selection pressures, creating feedback loops that lead to multistability within the genome. This theory accounts for the diversity of codon usage biases, a phenomenon for which formerly accepted mechanisms did not account.

2.2.2. Types of codon usage bias

The bias in any particular organism may be affected by some or all of several factors in varying degrees; it is the combination of these effects that accounts for the selective pressure on codon usage in every organism. It was initially assumed that biased usage was the result of selection for translational efficiency alone, but later work suggested that other factors also play a significant role.

2.2.2.1. Translational efficiency

Translational efficiency was the first theory formulated as an explanation for biased codon usage. Early research found a close correlation between an organism's choice of preferred codons and its population of isoaccepting tRNAs (Ikemura 1981), and observed that this would facilitate the translation of proteins whose genes use these codons by ensuring a constant, ready supply of the biomolecules (namely, the tRNAs) used during the translation process. Several researchers also confirmed that genes that are highly expressed (synthesized often) tend to use mostly preferred codons, while less highly-expressed genes use preferred codons with a lower frequency (Grantham, Gautier et al.

1980), (Ikemura 1981), (Bennetzen and Hall 1982), (Gouy and Gautier 1982), (Ikemura 1985). Work by Varenne *et al* supported this theory by showing that transfer RNA availability had a significant effect on the speed of the translation process: the recruitment of an amino acid by its transfer RNA was the limiting step during translation (Varenne, Buc et al. 1984). This confirmed that a codon whose transfer RNA is readily available will be translated more quickly than a codon with a rare transfer RNA. It was concluded that highly-expressed genes contained a large proportion of preferred codons because these genes experience the highest degree of selective pressure to be produced more efficiently by the organism. Genes that are expressed less frequently are under less pressure, and thus contain fewer preferred codons.

2.2.2.2. GC Content

GC(AT)-content refers to the percentage of nucleotides that are guanine or cytosine (adenine or thymine) in a DNA sequence. For a double-stranded DNA molecule, nucleotide proportions follow Chargaff's Rule (Chargaff 1950):

$$\%A = \%T \ \ and \ \ \%G = \%C \tag{1}$$

Recall that complementary base pairing between the two strands of a DNA molecule pairs G's with C's and A's with T's; the proportions in Chargaff's rule are the result of this pairing.

GC-content has been shown to vary drastically between organisms (Sueoka 1962). In some organisms, GC-content is extreme to the extent that it completely dominates the genome's choice of codons. Organisms with an extreme GC-content (those in which GC >> AT or AT >> GC) are said to be strongly characterized by GC-content bias (or AT-content, if the bias is towards AT rather than GC). The biological reason for this has not been conclusively determined, but several observations have been made with regards to the types of organisms that display strong content bias. Moran noted that the genomes of obligate intracellular pathogens and symbionts were greatly reduced with regards to the size of the genome and the number of genes it contained, and observed that these genomes tended to have very low GC-content (Moran 2002). Rocha and Danchin supported Moran's findings in a paper that showed that the genomes of obligate intracellular organisms (including pathogens and symbionts) tend to be richer in AT's than in GC's (Rocha and Danchin 2002); they extended this trend to bacterial phages, which are also host-associated, and to plasmid DNA, which is non-essential, self-replicating, and is sometimes considered parasitic. This paper noted that GC nucleotides are metabolically more "expensive" than AT's, and proposed that high AT content could be the result of a scarcity of GC's and selection for the use of available resources. A report by Foerstner *et al* later drew a correlation between the environment of an organism and the GC-content of its genome (Foerstner, von Mering et al. 2005); organisms from a similar environment tend to have more similar GC-content than do organisms from the same phyla. This report concluded that environmental factors were the strongest influence on the GC-content of a genome.

Variations in the GC-content in the third nucleotide of the codons have also been noted (Lafay, Lloyd et al. 1999); GC3-content is another source of codon usage bias.

2.2.2.3. Strand-related bias

A relatively small number of organisms have genomes characterized by a strong strand-specific skew in codon usage. Lafay *et al* demonstrated that the genomes of *Borrelia burgdorferi* and *Treponema pallidum* have a significantly different pattern of codon usage on the leading versus the lagging strand of the chromosome (Lafay, Lloyd et al. 1999). This trend was strong enough that the primary influence on codon usage in both organisms was the orientation with respect to the origin of replication, to the exclusion of translational effects. Other organisms characterized by this type of bias have since been identified.

Lafay *et al* also noted that *Treponema pallidum* was strongly characterized by strand-specific differences in nucleotide base composition; the leading strand was GT-rich compared to the lagging strand. This type of bias is known as GC-skew.

2.2.3. Quantifying codon usage bias

The goal of methods for quantifying and representing biased codon usage is to indicate which codons are major within the genome. The development of such methods has led to two distinct approaches. Some methods use multivariate or statistical techniques to identify the codons that are most strongly preferred (major) in a genome. Other methods assign a weight to each codon, indicating its frequency of use relative to its synonyms. This section will detail the development of these methods in chronological order, along with the pros and cons of each.

2.2.3.1. Frequency of preferred codons

One of the first papers to explore the correlation between biased codon usage and efficiency of translation also proposed a measure of the expressivity of a gene (Ikemura 1981). The tendency of highly-expressed genes to use a set of preferred codons led to the formulation of an equation to determine a gene's frequency of use of preferred codons (Ikemura 1981).

$$FOP = \frac{number \quad of \quad optimal \quad codons}{total \quad number \quad of \quad codons \quad in \quad gene} \tag{2}$$

A codon is "optimal" if it meets criteria for translational efficiency.

2.2.3.2. Codon bias index

Soon after Ikemura's FOP measure was published, Bennetzen and Hall came up with a very similar measure (Bennetzen and Hall 1982). Like FOP, their codon bias index attempted to characterize the proportion of preferred codons in a gene, but their ratio also takes into account the number of codons that would appear in a gene if usage were completely random. CBI is calculated by taking the number of optimal codons in a gene minus the number of these codons that would be expected with random usage, divided by the number of codons in the gene.

2.2.3.3. Correspondence analysis

Correspondence analysis was used by Grantham *et al* in the work that originally drew the correlation between biased codon usage in a gene and that gene's level of expression (Grantham, Gautier et al. 1981). They found that projecting genes into the space defined by the first two principal components of their codon frequency data separated the genes into two distinct groups according to their level of expression.

2.2.3.4. P1 and P2 index

The methods discussed so far have been concerned only with the effects of gene expressivity. The P1/P2 index method developed by Gouy and Gautier takes into account another component of codon usage bias: the choice of nucleotide in the third position of the codon (referred to as GC3-content elsewhere in this survey) (Gouy and Gautier 1982). Their P1 index is similar to the previous methods in that it is strongly correlated with gene expressivity; for each gene, it is the number of isoaccepting tRNA's for each codon weighted by the relative frequency of the codon in the gene. The P2 index is based on the strength of the codon-anticodon interaction between the mRNA template and tRNA. It is the frequency of "right choices" for the third nucleotide in a codon (the position which is most often degenerate).

2.2.3.5. Codon preference bias

Unlike the methods discussed so far, the codon preference bias does not require *a priori* knowledge of an organism's tRNA population (McLachlan, Staden et al. 1984). This method computes the probability of a gene's codon frequency given the amino acid composition of the organism's proteins, and uses a multinomial distribution to determine the probability of deviation from an "expected" frequency based on completely random usage. The expected frequency for a codon f_c was calculated as follows, where A_s is the usage of an amino acid A in sequence s and A has d_s codons, all equally used:

$$f_c = A_s / d_s \tag{3}$$

The results of this method were "well correlated" with the results achieved by the previous methods.

2.2.3.6. Cluster analysis

This method is significant in that it introduces the idea of a relative synonymous codon usage measure that indicates each codon's frequency of use relative to its synonyms, where previous methods have been concerned only with a genome's set of preferred codons. This measure will be used in several subsequently-developed methods, and is calculated by dividing the number of occurrences of a codon by the sum of the number of occurrences of its synonyms, as shown below (Sharp, Tuohy et al. 1986).

$$RSCU_{ij} = \frac{X_{ij}}{\frac{1}{n_i} \sum_j X_{ij}} \tag{4}$$

Sharp *et al* showed that yeast genes represented by RSCU vectors could be clustered into two groups, one containing highly-expressed genes and the other containing genes that are not highly-expressed. They also used a chi-squared statistic to calculate the bias levels of the genes, where bias level is defined as the difference between the usage of a codon in the gene and the average usage of the codon across the genome.

$$\chi^2 = \sum_{i=1}^{64} \frac{\left(CU_i - \overline{CU}_i\right)^2}{\sigma_i^2} \tag{5}$$

2.2.3.7. Codon adaptation index

Soon after Sharp *et al* developed the idea of a RSCU vector, Sharp and Li incorporated this measure into their Codon Adaptation Index (CAI) measure of synonymous codon usage bias (Sharp and Li 1987). Thus far, measures of codon usage bias have shared several limitations; the CAI measure was designed in response to these. Previous measures had only been able to assign a binary status to a codon; either a codon is optimal or it is not, with no opportunity to identify a degree of optimality. It was also impossible to perform a meaningful comparison of the biases of two different organisms because different organisms had different proportions of optimal to non-optimal codons. Sharp and Li's method addressed these issues by computing a vector based on the RSCU value discussed above; the vector is normalized to enable inter-species comparison.

The CAI method requires a list of highly-expressed genes; a weight for each codon based on its RSCU value is calculated using this set of genes as a reference set. The weight is the ratio of that codon's RSCU to the RSCU of its maximal sibling (the most frequently used codon).

$$w_{ij} = \frac{RSCU_{ij}}{RSCU_{i\max}} = \frac{X_{ij}}{X_{i\max}} \tag{6}$$

The weights for each codon are then used to compute CAI values for each gene. A gene's CAI value is the geometric mean of the weights for the codons in the gene.

$$CAI = \left(\prod_{k=1}^{L} w_k \right)^{1/L} \tag{7}$$

Next, the CAI values can be used to predict the level of expression of the genes not included in the reference set. It is important to note that the quality of this prediction depends entirely on the genes included in the reference set, as all the calculations are based on the codon usage in these genes.

2.2.3.8. Scaled χ^2

A previous measure of codon bias, cluster analysis, used a χ^2 metric to examine the deviation of observed codon usage in a gene from the expected usage (the average usage across the genome). Shields *et al* observed that these values were highly correlated with gene length, and introduced the scaled χ^2 measure to address this issue (Shields, Sharp et al. 1988). The χ^2 values are scaled by the number of codons in the gene.

$$\chi^2 = \sum_{i=1}^{64} \frac{\left(CU_i - \overline{CU}_i\right)^2}{\sigma_i^2} \tag{8}$$

$$\chi^2_{scaled} = \frac{\chi^2}{N_{codons}} \tag{9}$$

This method is intended to produce values indicating the level of bias in each gene.

2.2.3.9. Effective number of codons

The goal of the effective number of codons (N_c) measure was to calculate how much the codon usage of a gene differs from the equal usage of synonymous codons (Wright 1990). The benefits of this measure are that it can be calculated from sequence data alone, and is inherently independent of both gene length and amino acid composition, requiring none of the additional normalization that has been necessary for some of the previous methods. N_c values can range from 20 to 61; a value of 20 indicates that one codon is preferred to the exclusion of all synonyms for each amino acid, while a value of 61 indicates equal usage of all amino acid-codon codons (only stop codons are excluded).

2.2.3.10. Intrinsic codon deviation index

So far, one of the primary weaknesses of many measures of codon usage bias is the requirement of *a priori* knowledge, either of tRNA levels or the expression rates of at least some of the genes to which the measure is applied. The measures with this requirement are thus less useful for studying genomes about which little information is available. Freire-Picos *et al* developed the intrinsic codon deviation index (ICDI) to address this weakness (Freire-Picos, Gonzalez-Siso et al. 1994). It is calculated in a two-step process; first an index for each amino acid is calculated based on the RSCU values of its codons, then the individual indices are summed to obtain the final ICDI.

$$S_k = \sum_{i=1}^{k} \frac{(n_i - 1)^2}{k\,(k - 1)} \tag{10}$$

$$ICDI = \frac{\sum S_2 + S_3 + \sum S_4 + \sum S_6}{18} \tag{11}$$

ICDI gives values ranging from 0 to 1; higher values indicate a stronger bias.

2.2.3.11. Major codon usage

Major codon usage was a technique developed by Kanaya *et al* to aid in the study of how codon usage relates to tRNA abundance and gene expressivity (Kanaya, Yamada et al. 1999). A gene's MCU is determined by dividing its number of major codons by the total number of codons in the gene. Major codons are identified via multivariate analysis of a matrix consisting of RSCU vectors for each of the genes in a genome. The first principal component of this matrix is extracted using PCA; each gene's RSCU vector is projected along the first principal component, resulting in a one-dimensional vector describing codon usage in that gene. Each codon is then examined to determine whether it contributes positively to the general trend in the projection. Codons that do contribute positively are considered major, and used to compute MCU.

2.2.3.12. Self-consistent codon index

In 2003, Carbone *et al* introduced a variation on Sharp and Li's CAI method (Carbone, Zinovyev et al. 2003). The new measure was originally also called CAI, but the authors later requested that it be referred to as the self-consistent codon index (SCCI) to avoid confusion. The SCCI method diverges significantly from all previous methods. Where previous methods have focused solely on the concepts of translational efficiency bias and computing gene expression levels, the SCCI method seeks to identify the dominating bias in the genome regardless of source and rank the genes according to this bias. The dominant bias can then be identified, and the genome labeled by whichever type of bias most strongly characterizes it.

The SCCI measure is very similar to CAI in that it uses the same method for calculating codon weights and gene indices (see Equations 6 & 7). Where it differs is the way in which the reference set is selected. CAI uses a set of known highly-expressed genes; SCCI finds its reference set through an iterative search of the genome. Each iteration selects the genes that adhere the most strongly to their own bias (the most self-consistent genes) and computes weights based on these genes for the next iteration. The method will be discussed in greater detail in Section 3.2.3.

2.2.3.13. Modified self-consistent codon index

As mentioned in the previous section, SCCI does not search specifically for translational efficiency bias, searching instead for the most strongly self-consistent bias. Raiford *et al* introduced the modified SCCI method to use the same iterative search to look specifically for genes characterized by translational efficiency bias (Raiford, Krane et al. 2008). Modified SCCI directs the search for the reference set away from genes with extreme GC-content, which is the greatest confounding factor of translational efficiency.

3. Exploration of codon usage bias trends in free-living and intracellular prokaryotes

3.1. Introduction

It has been observed since the first study of codon usage bias that each organism has a unique pattern of codon usage. Carbone *et al* made use of this observation along with their SCCI measure of codon usage to formulate the concept of a codon usage space, a 64-dimensional space where each organism is represented by its 64-dimensional vector of codon weights calculated via SCCI (Carbone, Kepes et al. 2005). Spatial proximity in this space is a function of biological similarity; organisms with similar biological characteristics will be closer in codon usage space than will more dissimilar organisms.

The validity of this concept was tested and proved on a limited number of biological traits: lifestyle (thermophilic vs. mesophilic) and respiration type (aerobic, anaerobic, facultative aerobic, facultative anaerobic). Organisms were also separable in codon usage space according to the type of bias that most strongly characterized their genome (referred to in the paper as its signature). Although the results obtained thus far are consistently encouraging with regards to the usefulness of codon usage space as a tool for classifying and comparing organisms, the concept has been tested on a relatively small number of biological traits. This section of the thesis will attempt to further validate this concept by applying this methodology to previously unexplored types of organisms to determine whether this trend generalizes to other biological characteristics.

3.2. Materials and methods

3.2.1. Selecting an appropriate comparison

A good deal of biological evidence suggests that the codon usage of obligate intracellular prokaryotes may differ sufficiently from that of more free-living prokaryotes to make a comparison between organisms of these two types an acceptable candidate for this exploration. Research has shown that the GC-content of the genomes of obligate intracellular pathogens and symbionts differs from that of more free-living bacteria (Moran 2002; Rocha and Danchin 2002), and Foerstner *et al* (Foerstner, von Mering et al. 2005) demonstrated that the GC-content of bacterial genomes tends to vary with the environment to which the organism is adapted. The reasons for these variations have not been conclusively determined, but GC-content is a known cause of codon usage bias; so it is possible that there may be some greater distinguishing factor in the codon usage of these two types of organisms that can be detected via codon usage space.

3.2.2. Acquisition and classification of genomic data

Complete genome sequences for forty prokaryotic organisms are analyzed to determine relative codon usage frequencies with regards to their dominant bias (see Table 2). Each organism is labeled as either intracellular or free-living. The intracellular classification includes obligate intracellular parasites and symbionts; organisms that are not obligatorily intracellular are classified as free-living. This determination is based on the organism's entry in the Entrez Genome Project, which lists an organism's environment as terrestrial, aquatic, multiple, host-associated, or specialized. Terrestrial, aquatic, and multiple-environment organisms are considered free-living, while host-associated and specialized organisms are further investigated to determine the appropriate classification. Sufficient information to classify each organism is found in the organisms' descriptions in Entrez, along with each organism's genomic GC content, which will be utilized later in this section. Genome sequences are obtained from the Genbank annotated files for these organisms (as of October 2008). All sequences labeled as genes are included. Sequences from plasmid DNA are excluded to remove concerns that plasmids may have significantly different codon usage than chromosomal DNA and therefore skew the results away from the genome's native usage (Rocha and Danchin 2002).

Table 2. List of organisms

Organism Name	Group	NCBI habitat
Acholeplasma laidlawii	Social	Specialized
Aeromonas salmonicida	Social	Aquatic
Anaplasma marginale	Intracellular	Host-associated
Bartonella bacilliformis	Intracellular	Host-associated
Baumannia cicadellinicola	Intracellular	Host-associated
Bdellovibrio bacteriovorus	Social	Multiple
Blochmannia floridanus	Intracellular	Specialized
Borrelia burgdorferi	Intracellular	Host-associated
Bacillus subtilis	Social	Terrestrial
Buchnera aphidicola	Intracellular	Host-associated
Chlamydia trachomatis	Intracellular	Host-associated
Clostridium perfringens	Social	Multiple
Ehrlichia ruminantium	Intracellular	Host-associated
Haemophilus influenzae	Intracellular	Host-associated
Lactobacillus plantarum	Social	Host-associated
Lactococcus lactis	Social	Multiple
Lawsonia intracellularis	Intracellular	Host-associated
Listeria innocua	Social	Multiple
Mesoplasma florum	Intracellular	Host-associated
Methylobacillus flagellates	Social	Specialized
Mycobacterium smegmatis	Social	Host-associated
Mycoplasma pulmonis	Intracellular	Host-associated
Nanoarchaeum equitans	Intracellular	Host-associated
Onion yellows phytoplasma	Intracellular	Host-associated
Orientia tsutsugamushi Ikeda	Intracellular	Host-associated
Polynucleobacter necessaries	Intracellular	Host-associated
Prochlorococcus marinus	Social	Aquatic
Pseudomonas aeruginosa	Social	Multiple
Ralstonia solanacearum	Social	Multiple
Rickettsia felis	Intracellular	Host-associated
Saccharopolyspora erythraea	Social	Terrestrial
Salmonella enterica	Social	Multiple
Sorangium cellulosum	Social	Terrestrial
Staphylococcus aureus	Social	Host-associated
Synechococcus sp. WH 8102	Social	Aquatic
Thermus thermophilus	Social	Specialized
Wigglesworthia glossinidia	Intracellular	Host-associated
Wolbachia endosymbiont of Brugia malayi	Intracellular	Host-associated
Xanthomonas oryzae	Intracellular	Host-associated
Yersinia pestis	Social	Multiple

Genbank annotated files are text files that contain a great deal of information in addition to the gene sequences required for this research. A PERL script developed by Raiford (Raiford 2005) is used to parse the files and extract gene names and sequences (see

Appendix B). The script is invoked using a command with the following format and parameters.

```
 perl getGenes.pl -noeq -nothree -nophage -len X FILENAME      (12)
```

The *–noeq* and *–nothree* flags exclude genes whose nucleotide sequence does not translate to the amino acid sequence given in the file and genes whose nucleotide sequence is not divisible by three, respectively. The *–nophage* flag removes genes whose annotations indicate they may be the result of horizontal gene transfer, and *–len X* allows genes whose nucleotide sequence has fewer than *X* characters to be disregarded. An initial subset of organisms was parsed with and without the *–noeq* and *–nothree* flags to determine the impact and frequency of such errors in the annotated files; instances of genes meeting these criteria were relatively few, so it is possible to cull these genes without significantly reducing the amount of genomic data available. The full set of genomes is processed with the *–noeq* and *–nothree* flags (excluding the erroneous genes). Genes are not culled on the basis of length or phage association; the length of a gene has no apparent effect and phage-associated genes are extremely unlikely to have any measurable impact on the genome's dominant bias.

The output from the PERL parsing script consists of a list of genes for every organism, containing the gene's name and nucleotide sequence. This data allows the codon usage to be calculated for each genome.

3.2.3. Calculating the dominant bias

Codon usage space is defined by 64 dimensions; each dimension corresponds to the usage of a single codon. An organism's coordinates in this space are the codon weights calculated from the reference set determined by the SCCI algorithm (Carbone, Zinovyev et al. 2003); the location of an organism is thus dependent on the codon usage of its genome's dominant bias. The original work done with codon usage space used a 64-dimensional space, but five of these dimensions are inherently meaningless with regards to codon usage bias; bias can only be displayed by protein-coding synonyms, so stop codons and the codons for methionine and tryptophan (which have only one codon each) do not add any meaningful information to the space. The work in this thesis uses only the 59 informative dimensions.

Each iteration of the SCCI algorithm calculates a codon weight vector based upon the relative frequency of codon usage in a reference set. Each codon's weight w is that codon's count X in the reference set divided by the count of its maximal sibling (the synonymous codon that appears in the reference set the most often).

$$w_{ij} = \frac{X_{ij}}{X_{i\max}} \tag{13}$$

The weight w_{ij} and count X_{ij} refer to the jth synonymous codon for the ith amino acid. Maximal synonyms have weights of 1.0 (their count is divided by itself); in the weight vector, each amino acid will have at least one codon with a weight of 1.0. These are the major (most strongly preferred) codons. Non-major codons have weights in the range of [0, 1), where smaller weights indicate less frequent usage relative to the major sibling. The larger the weight, the more preferred the codon. The SCCI value for each gene is calculated by taking the geometric mean of the weights of the codons found in the gene.

$$SCCI = \left(\prod_{k=1}^{L} w_k \right)^{1/L} \tag{14}$$

The number of synonymous, protein-coding codons in the gene is *L* (STOP codons and the codons for methionine and tryptophan are disregarded, as mentioned above). A gene's SCCI is therefore dependent on the majority of its codons; a large number of preferred codons results in a high SCCI and fewer or less strongly preferred codons result in a lower SCCI. Next, the genes are sorted by their SCCI values. The *n*/2 genes with the highest values become the reference set to be used in the next iteration, where *n* is the number of genes in the current reference set. The algorithm terminates when the reference set converges and contains approximately 1% of the genome's total number of genes. SCCI values are dependent on the codon usage in the reference set, so a gene in the reference set generally has a higher SCCI than a gene outside the reference set. Because each iteration removes the genes with the lowest SCCI values from the reference set, the end result is a set of genes that contain a large number of their own major codons. That is, these genes have the most strongly self-referential codon usage in the genome. This is the meaning of the "dominant" bias.

In the first iteration of the algorithm, the reference set is initialized to contain the entire genome. It follows that the codon weights obtained in the first iteration represent the background, or average, codon usage in the genome. While not necessary with regards to classifying an organism via codon usage space, the idea of a genome's average codon usage will be utilized later in this thesis.

The SCCI algorithm is implemented in Ruby, a dynamically-typed object oriented scripting language that incorporates aspects of functional programming. Ruby's flexibility makes it an ideal choice for this project. The abilities to modify the behavior of built-in functions and add new methods to built-in classes allow the language to be customized to meet the needs of this particular problem domain, the processing of genomic data. For example, the following method is added as a String class method to facilitate the processing of DNA sequences, and makes use of the Ruby feature that allows blocks of code to be passed to an iterator via the *yield* keyword, a feature that borrows from the functional programming style.

```
class String
  def each_codon
    position = 0
    while position <= self.length-3
      yield self[position, 3]
      position = position + 3
    end
  end
end
```

(15)

The method defines an iterator that takes a block of code as a parameter and applies the block to every consecutive three-character substring, which corresponds to a codon when applied to a string that represents a DNA sequence. The iterator is invoked using the following syntax.

```
geneSeq = String.new
# initialize geneSeq
…
geneSeq.each_codon{
  |cdn|
  # process codon
  …
}
```

(16)

The implementation of the SCCI algorithm reads the text files containing the gene sequence data, and computes a codon weight vector for each of the forty organisms under study. Although these vectors are sufficient to assign each organism a point in codon usage space, the space is so highly-dimensional to be impossible to visualize in this form. The application of unsupervised linear projection will reduce the dimensionality so that any trends in codon usage variation between the groups may be explored.

3.2.4. PCA

Principal components analysis (PCA) (Hotelling 1933), (Smith 2002) is an unsupervised pattern recognition and visualization technique. It is a useful tool for the visualization of multivariate data sets, and has been used previously in codon usage bias research. Grantham *et al* used it to separate highly- from lowly-expressed genes according to the genes' codon frequencies (Grantham, Gautier et al. 1981), and it was applied to the weight vectors of some of the organisms in the original examination of codon usage space in order to visualize the organisms' relative positions (Carbone, Kepes et al. 2005), a similar use to what is required here.

PCA can be performed on any data which may be placed in an *n*-by-*d* matrix, where *n* is the number of data points and *d* is the number of dimensions. The principal components of such a matrix are computed by first calculating the data's covariance and placing the results in an *n*-by-*n* covariance matrix where each element represents the covariance between two of the dimensions. Next, the eigenvectors of the covariance matrix are calculated. An eigenvector of a matrix is a vector that, when multiplied by the matrix, results in a vector that is a scalar multiple of the eigenvector. The scalar is the eigenvector's eigenvalue. In the case of principal components analysis, the eigenvectors are the principal components of the data set; they represent the directions of greatest variance within the data. A data set with *d* dimensions will have *d* orthogonal principal components. Ordering the principal components by descending eigenvalue ranks them according to the amount of the total variance each component explains and allows the first principal components to be identified. Often the first two or three principal components contain the majority (~80%) of the variance in the data, allowing high-dimensional data to be projected into this space with only a small loss in the amount of total variance. Dimensionality reduction is accomplished in this manner. The computations necessary to performing PCA on the codon weight vector data were done in the MATLAB environment due to the ease of matrix computations and data plotting in this language.

The codon weight vectors computed using the method described above are placed in a 40-by-59-dimensional matrix, where each row corresponds to an organism and each column corresponds to a codon (one dimension of the codon usage space).

$$\begin{array}{ccccc} & c_1 & c_2 & \cdots & c_{59} \\ o_1 & w_{1,1} & w_{1,2} & \cdots & w_{1,59} \\ o_2 & w_{2,1} & w_{2,2} & \cdots & w_{2,59} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ o_{40} & w_{40,1} & w_{40,2} & \cdots & w_{40,59} \end{array} \tag{17}$$

The MATLAB programming environment is then used to perform principal component analysis on this matrix. The first principal component accounts for 64% of the data's variance and the second contains an additional 12%, ensuring that a projection of the data into these dimensions is a reasonable representation of the spatial relationships in the original 59 dimensions. The principal components are used to project the original data into the plane formed by the first two principal axes, thus reducing the dimensionality of the data to two dimensions as shown in Figure 4. To verify that any patterns shown in the data are not artifacts of a small subset of the genomes, this analysis is performed on the full data set as well as several subsets generated by holding out a random 50% of the genomes from both classes. Every projection of a hold-out data set shows a similar pattern to that displayed by the entire data set, verifying that the trends therein persist among the full set of genomes. The hold-out analysis was also done in MATLAB.

The projection of the codon weight vectors of the forty organisms into principal-component space is an accurate representation of the spatial relationships of the organisms in codon usage space, with a dimensionality that allows the data to be visually examined. Any trends in the data can now be examined and analyzed, towards the end of determining their biological cause.

3.2.5. Exploration of computational properties of codon usage

The data analysis described above shows a clear separation between free-living and obligate intracellular prokaryotes in codon usage space. Because this separation results from patterns of codon usage that differ between the two groups, it is possible that it can be duplicated using computational properties of the organisms' codon usage rather than the full codon usage vectors. While clearly informative, the 59-dimensional codon vectors do not offer any insight or intuition with regards to the genomic characteristics responsible for the arrangement of organisms in codon usage space, and organisms can only be studied in relation to other organisms that have been characterized the same way. If some mathematical property or set of properties derived from codon weight vectors captures the same biological relationships as do the full weight vectors, it may provide some information regarding biological or genomic characteristics that is not provided by the codon weight vector alone, or at least allow the source of differences between organisms to be easily identified.

The mathematical methods used and the biological properties that each one is intended to represent are summarized in Table 3.

Table 3. Summary of mathematical properties of codon weight vectors

Method #	*Intended Biological Property*	*Computations*
1	The balance between highly-biased and weakly-biased synonymous codon groups	The mean of squares of the mean of squared weights in each synonymous group
2	The average degree to which each synonymous codon group is biased	The standard deviation of each synonymous group's codon weights multiplied by the number of synonyms, averaged over all groups
3	Relative bias levels of all synonymous groups	Method # is calculated for all synonymous groups; these values are sorted, divided into sixths, and the average taken of each sixth
4	Average level of adherence in the genome	Average of all SCCI values in the genome
5	The degree to which adherence varies throughout the genome	The standard deviation of the genome's SCCI values

The mathematical properties listed above are calculated for each of the organisms under study and the results placed in a 40-by-10-dimensional data matrix. Although this matrix is of much smaller dimensionality than the matrix containing the codon weight vector data, it is still too highly dimensional to visualize without additional analysis. Several pattern recognition techniques are applied to this matrix to determine whether it captures similar patterns to those present in the representation of the organisms in codon usage space. A PCA plot was generated using the same method described in Section 3.2.4 and shows no separation between the two classes (Figure 3). Similar results are obtained using Linear Discriminant Analysis (LDA). The lack of class separation in the PCA and

LDA results rules out the possibility that the classes are linearly separable using dimensionality reduction. However, the data may be separable using the full data matrix, and the classes may be non-linearly separable, two possibilities that PCA or LDA would not necessarily reveal. To test this hypothesis, a k-Nearest-Neighbors classifier is applied to the original 40-by-10 dimensional matrix containing the raw results of the computations on the codon weight vectors. Similar classification accuracy obtained by both the vector properties and the original vectors would indicate that the mathematical properties of codon weight vectors may be sufficient to represent to arrangement of organisms in codon usage space. The kNN classifier reached an accuracy of 70.75% with k=6 when applied to the mathematical-property data, while the same classifier applied to the codon weight vector data reached an accuracy of 85.00% (k=5). These results indicate that the biological information contained in the codon weight vectors is an artifact of the entire vector, and is reduced by applying the computations described in this section.

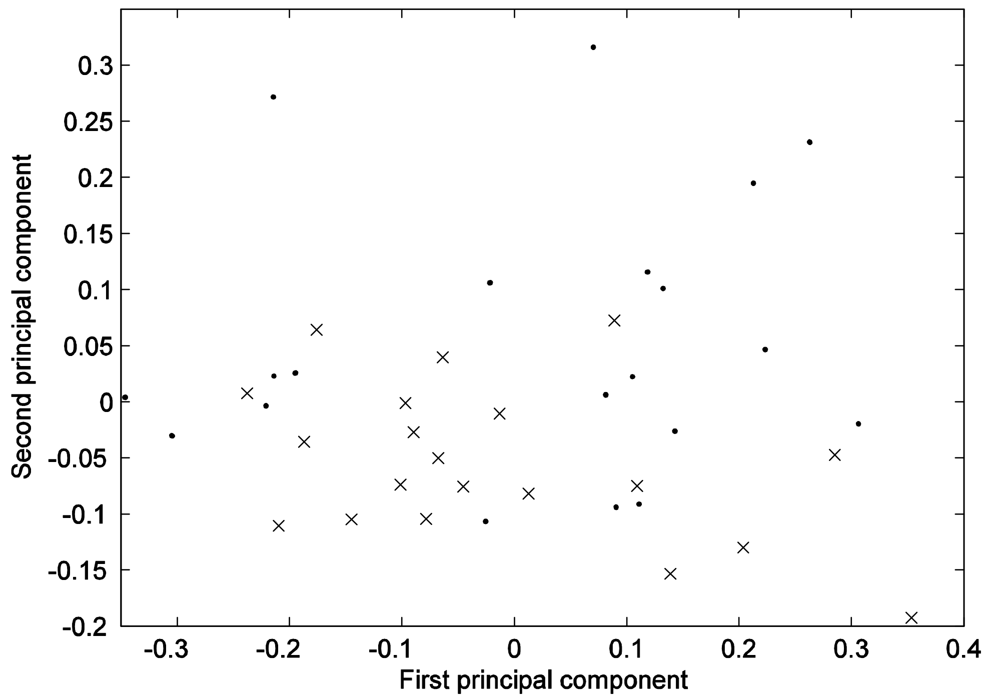

Figure 3. Organisms represented by mathematical properties of codon usage bias in principal components space

3.2.6. Deducing the meaning of the principal components

A comparison between free-living and intracellular organisms was originally chosen because evidence in the literature suggests that these types of organisms differ inherently in their choice of codons. Specifically, free-living organisms are thought to select for a higher GC content than do intracellular organisms. Because of this known differentiating factor between the two classes, it will be beneficial to determine whether any variance between the two groups is solely a result of differing GC content. This will indicate the presence or absence of any other biological factors that might affect the arrangement of the two groups in codon usage space. Each organism's genomic GC content was obtained when the organisms' annotated files were retrieved from Genbank. This data is now incorporated into the plot obtained in the previous section to determine the effect of GC content on the organisms' spatial arrangement in the plot.

The GC-content values for these genomes are percentages ranging from ~25% to ~75%. To determine whether a correlation exists between the organisms' arrangement in codon usage space and their GC content, the GC-percentage values are divided into quartiles and each organism's location on the plot is assigned a symbol indicating to which quartile it belonged. The resulting plot shown in Figure 5 indicates a definite gradient of GC-content across the first principal component, but there is no apparent correlation between GC-content and the second principal component. To enumerate the strength of the correlation between the principal components and GC content, a Pearson's correlation coefficient was calculated between these values. The correlation between the genomic GC content and position along the first principal component was very strong (R = 0.96), but there was no correlation between GC content and position along the second principal component (R = 0.03). These correlations are consistent with the trends shown on the PCA plot.

3.3. Results

Figure 4 shows that the genomic data for the forty organisms form three distinct clusters in codon usage space, as visualized by PCA and projection along the first two principal components. The cluster in the fourth quadrant of the plot contains primarily intracellular organisms, with the remaining two comprised mostly of free-living organisms. The fact that the intracellular organisms are so clearly separated from the free-living organisms supports the hypothesis that proximity in codon usage space is an accurate indicator of biological similarity, which was the primary goal of this chapter of the thesis.

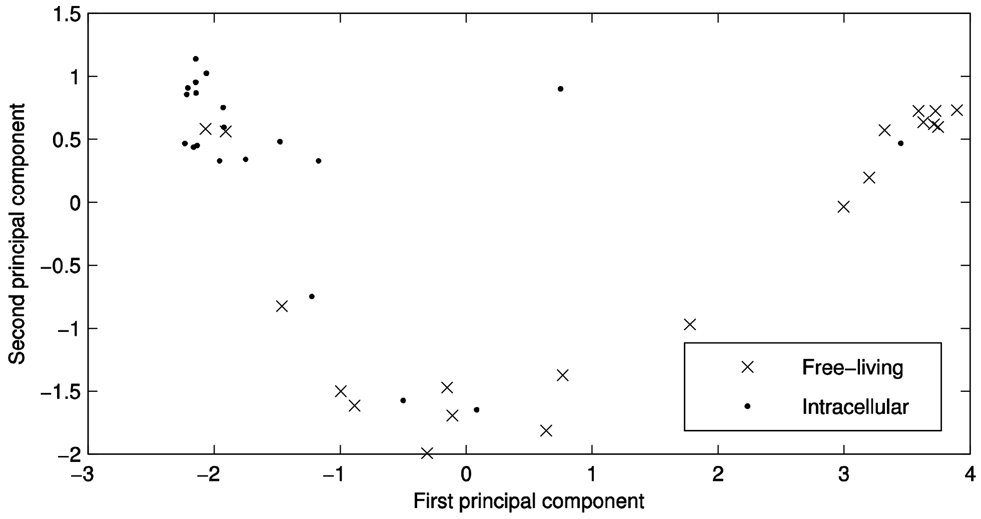

Figure 4. Projection of genomes in codon usage space into principal component space

To determine whether the separation shown in Figure 4 is an artifact only of the organisms' GC content, GC-content data for the organisms are added to the PCA plot and correlation coefficients are calculated between the organisms' GC content and their positions along the first and second principal components. Figure 5 indicates a gradient of GC content along the first principal component; the values for both appear to increase together. The Pearson's correlation coefficient between GC content and location on the first principal component is high (R = 0.96), confirming the relationship shown in the figure. However, neither the correlation coefficient nor the data plot shows any relationship between GC content and the second principal component. Although the first principal component (and therefore, GC content) is responsible for most of the separation between the two classes in this space, about half of the free-living organisms are separated from the other half of the free-living and all of the intracellular organisms along the second principal component as well as the first, indicating the presence of some other influence on the organisms' positioning in this space. That is, some element of codon usage other than GC content differs significantly among this group of organisms.

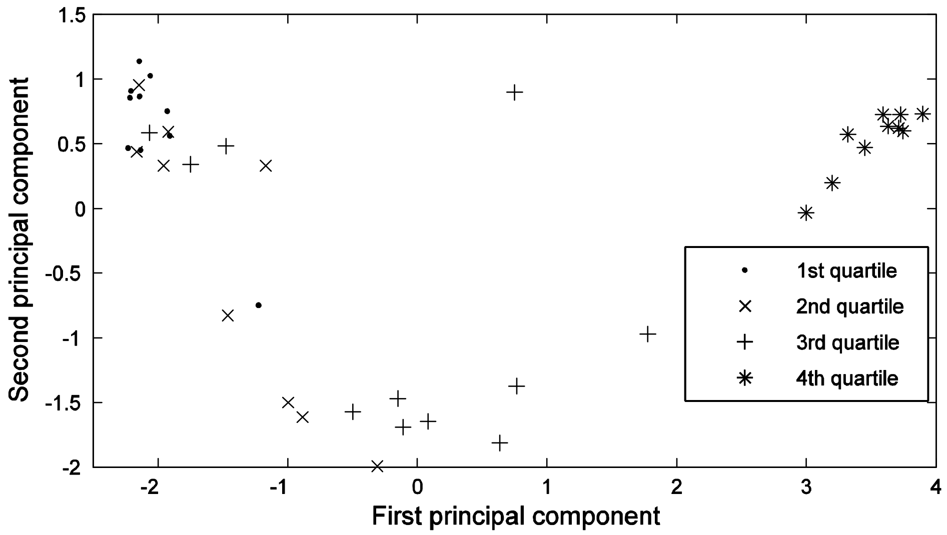

Figure 5. Genomes in PC space, labeled by GC content

There are several possible explanations for the presence and placement of two separate groups of free-living organisms. The first takes an evolutionary approach; these organisms may have only recently (from an evolutionary standpoint) adopted their free-living lifestyle, and their codon usage is still changing to adapt to their new environment. In this case, the central cluster may be a kind of midpoint between two class clusters, and the genomes therein may be at different points along the transition. This possibility accounts for the dispersion of the cluster along the first principal component fairly well as the group of intermediate organisms falls between the other two in this dimension, but does not provide as good an explanation for the degree of separation along the second. If the trait that distinguishes these genomes from the rest is simply that their codon usage is in transition, a more gradual separation would be expected rather than the relatively wide gap that is shown.

Alternatively, the genomes in the third cluster may represent an entirely separate biological group, distinguished from the rest by GC content as well as some other, unknown trait. Some investigation into the biological basis for the second principal component would shed some light on this possibility.

The results of this chapter indicate that free-living organisms are separable from intracellular in codon usage space, and prove that the concept of codon usage space as a representation of biological similarity is valid for a comparison of these two groups. GC content is primarily responsible for the separation between the two classes, which is also consistent with previous results. However, the representation of these organisms in codon usage space indicates the presence of some biological factor other than GC content that causes additional differentiation among the organisms. Because codon usage space is heavily dependent on codon usage bias to identify biological similarities, metrics that measure and allow comparisons among differently biased patterns of codon usage may be helpful in identifying this unknown differentiating factor.

4. Computing the strength of codon usage bias

4.1. Introduction

From its beginning, one of the goals of codon usage bias research has been to develop methods by which biological information can be obtained computationally from genomic sequence data. Such information obtained by recently-developed methods such as CAI and SCCI consists only of codon usage as defined by the dominant bias in the genome. Most techniques introduced prior to CAI were aimed primarily at translational bias. These often attempted to characterize the codon usage in a genome in such a way that the degree of the genome's tendency towards biased usage could be determined, addressing the question of what the codon usage bias in a genome looked like as well as the question of the strength of the bias. Knowing how strongly a genome is biased could be useful information for the characterization and comparison of organisms, but these early metrics were computed in such a way that comparisons among organisms were not possible. CAI, SCCI, and similar algorithms have solved part of the problem; codon usage can now be evaluated so that inter-species comparisons are meaningful and informative. If a measure of bias strength could be derived from these procedures, it may provide a useful new tool for genomic study and comparison.

Current literature contains few means for the measurement of the strength of biased codon usage; one is a population genetics model developed by Sharp *et al* (Sharp, Bailes et al. 2005). It addresses the problem of comparing selected codon usage bias across species, but has several limitations. Due to the nature of the computations, it only utilizes four of the eighteen amino acids able to display biased usage of their synonymous codons, so a great deal of the information in the genome is not taken into account. The model is also intended to measure only the strength of codon usage bias caused by selection for translational efficiency, as opposed to bias caused by mutation or other effects. While the population genetics model is useful within its scope, a broader evaluation may provide a more universally useful tool. Carbone *et al.* used a system of mathematical criteria and thresholds to determine a genome's tendency towards different types of bias (Carbone, Kepes et al. 2005), but no thresholds were provided to separate strong tendencies from weak tendencies except in the case of translational bias.

SCCI and related tools for the study of codon usage bias have attempted only to characterize a genome's bias so that the causes of the biased usage can be determined; they are not intended to compute strength or any other data regarding properties of the bias other than the codon usage therein. However, SCCI depends on the idea of a "dominant" bias, a set of genes containing a "family of codons that appear in most genes with the highest frequency" (Carbone, Zinovyev et al. 2003). It is unclear whether this definition is also sufficient for determining the strength of the bias or whether it better describes some other property of a genome's codon usage.

This chapter of the thesis addresses these issues by clarifying the existing terminology associated with bias strength and formulating a concrete definition of the term. With this definition in mind, current measures of codon usage bias are evaluated with regards to their fitness to be used as or developed into a bias strength metric. A new metric is proposed that computes bias strength according to the criteria laid out in the disambiguated definition and this metric is evaluated along with existing measures. The purpose of the evaluation is twofold; the best measure of bias strength is determined and the measures are also evaluated as a group to determine whether there is a subset that provides meaningful, unique information about the genome. Such a subset may constitute a useful set of tools to augment the study of codon usage bias.

4.2. Materials and methods

This section utilizes the same codon usage bias data for the same forty organisms discussed in the previous chapter, computed in the same manner. The test data for the metrics under evaluation thus consist of genomic sequences, a codon weight vector, and a reference set of genes for the forty organisms listed in Table 2.

4.2.1. Definition of bias strength

Measures of codon usage bias generally take one of two approaches. The first approach is in terms of individual genes; a gene is “very biased” if it meets some criteria defined by the technique in use. For example, according to the SCCI algorithm, a gene is “very biased” if its SCCI value is close to 1.0. A broader view is taken by methods following the second approach. These measures of bias examine the difference between the observed codon usage and some baseline, either the average usage in the genome or completely balanced usage. While the first approach may be helpful in terms of studying individual genes, one of the goals of this metric is to be able to apply it to the dominant bias in a genome, and the global approach is thus more useful. As the metric will be of limited use if it does not allow meaningful comparisons to be made among organisms, it will be preferable to calculate bias strength in terms of the difference from completely balanced usage rather than in terms of a genome's average codon or amino acid usage.

Bias strength can thus be defined as the difference between a genome's observed codon usage and equal usage of all synonymous codons. The degree to which a genome is strongly biased is the degree to which it prefers a small set of major codons over the alternative codons; the greater the preference across all synonymous groups, the stronger the bias. An effective measure of bias strength thus answers the question, "How preferred are the major codons?" By this criterion, a genome that displays the strongest possible bias will use one codon per amino acid, to the exclusion of all synonymous codons. The SCCI weight vector for such a genome will assign each major codon a weight of 1.0 and the remaining codons weights of 0. In terms of a bias strength metric, computing the metric on this vector should result in the value corresponding to the metric's upper bound. Equal synonymous codon usage is represented by a codon weight vector in which every weight is 1.0; this vector should result in the value of the metric's lower bound. Figure 6 illustrates, for a synonymous group of four codons, possible weight distributions displaying biased usage of differing strength. Note that the weight distribution labeled as being weakly biased represents codon usage that is relatively close to balanced, while the weights labeled as strongly biased represent codon usage closer to the maximum possible bias (preference of one codon over all others).

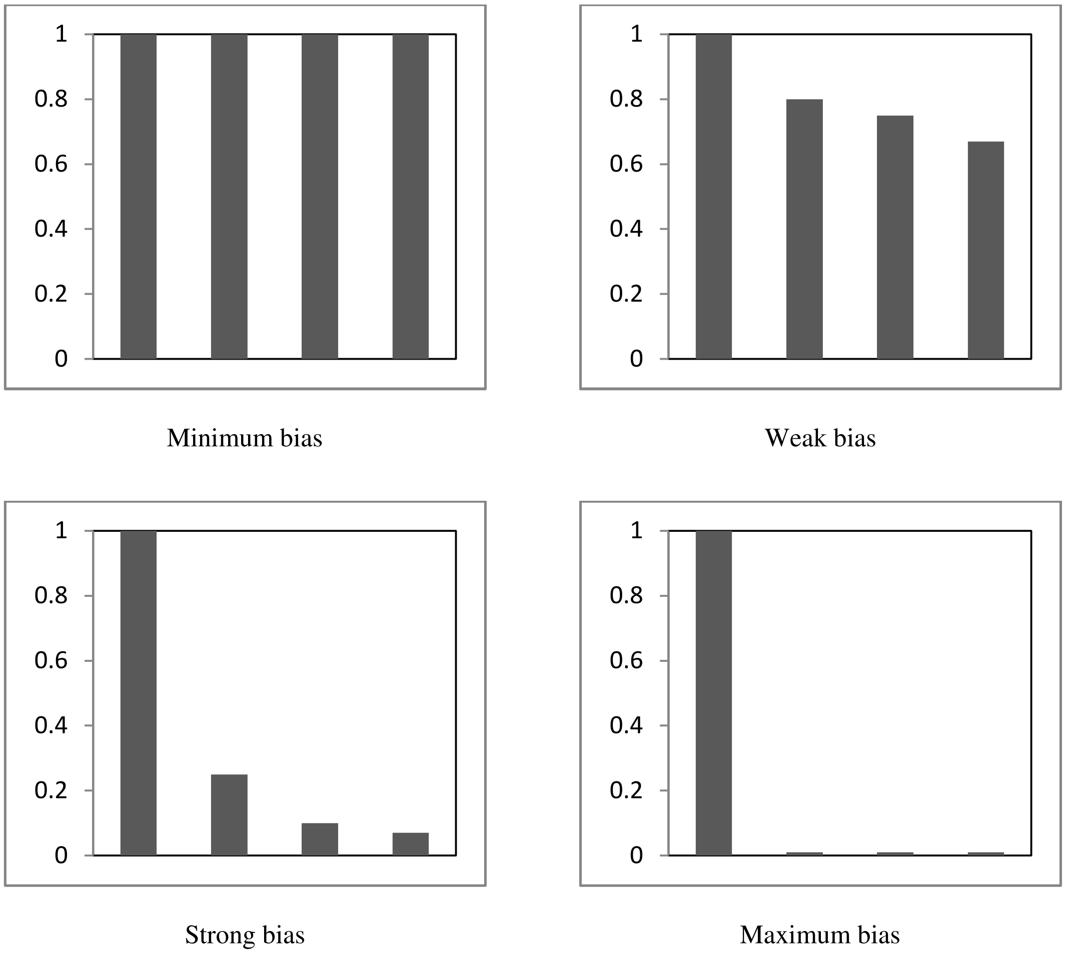

Figure 6. Bias strength examples

4.2.2. Properties of a bias

The formulation of codon usage bias by the SCCI algorithm allows for the definition of an additional property to the strength measure defined above, which is discussed here for the purposes of clarification and to distinguish between the two concepts. This property measures the extent to which a sequence contains a large proportion of codons that are frequent in the reference set. Because it has to do with how well a gene conforms to the codon usage defined in the reference set, this quality will be referred to as *adherence*.

The concept of adherence differs from the concept of strength in that adherence places more importance on how frequently the preferred codons appear in a sequence, rather than the degree to which these codons are preferred over others.

There are several possibilities for determining the level of adherence in a gene. The simplest is to examine the sequence's frequency of the bias' preferred codons relative to non-preferred codons. Another possibility is to calculate how closely the relative frequencies match for all synonyms between the sequence in question and the reference set. The difference between these two approaches can be clarified with a simple example. Consider the case of a bias consisting of a pair of synonymous codons where one codon *A* is used twice as often as codon *B*. Under the first approach, *A* is the major codon so the adherence score of a sequence increases with its use of codon *A*; sequences with a higher relative frequency of codon *A* to codon *B* have higher adherence because they use more of the major codon. By the second approach, the sequences with the highest adherence scores will be the ones that contain exactly twice as many codon *A*'s as codon *B*'s because that is how the codons are distributed in the reference set. Sequences that deviate from this distribution will have lower adherence.

A possible benefit of using the first approach is that the cases of minimum and maximum adherence scores are more clearly defined. The maximum adherence score will be assigned to sequences consisting entirely of major codons, and scores will decrease proportionally with the number of major codons in a sequence. Sequences with no major codons will be assigned the minimum adherence score. Using the second approach, the maximum adherence score is given to sequences whose codon distribution exactly matches that in the reference set, and decreases as the distribution varies. However, this does not account for the direction of a sequence's deviation from the reference set; adherence scores can descend along multiple paths, and the meaning of the minimum adherence score is not clear.

4.2.3. Examination of existing metrics

Although many measures of biased codon usage exist, most of them aim to characterize codon usage rather than determine the strength of any bias that is present. However, some of these measures may still be suitable for this purpose. An acceptable metric for bias strength meets the following criteria.

- It is based on the dominant bias, or general codon usage in the genome, rather than a specific type of bias.
- It is normalized in such a way that meaningful comparisons can be made between organisms.
- It indicates the degree to which the major codons are preferred over others in highly-biased genes, adhering to the definition of bias strength provided above. Indicating the degree to which major codons are preferred across the entire genome is less useful because the strength of the bias is diluted by genes which may be only weakly biased.

This section will discuss each of the measures described in Section 2.2.3 with regards to how well they meet these criteria.

The measures of codon usage bias that were developed early on are largely unsuitable for measuring bias strength. These include Frequency of Preferred Codons, the Codon Bias

Index, Correspondence Analysis, and the P1/P2 measure, all of which are designed to measure translational bias only. Most of these measures are also not normalized in such a way that allows meaningful comparisons among genomes with different amino acid or codon frequencies.

Codon preference bias is based on the organism's amino acid composition, which would make comparisons between organisms with different amino acid usage difficult, if not impossible. CBI also does not indicate the majority of codons, but rather the probability of their frequency given an amino acid composition. Very frequently-used codons would thus have a similarly high probability to very infrequently-used codons, rendering this type of measure useless for the purposes of bias strength, which is based on the difference between the two.

The RSCU measure introduced along with cluster analysis of codon frequency allows inter-species comparisons and gives information that is applicable to general codon usage rather than a specific bias, but it calculates this information over the entire genome rather than a set of highly-biased genes. Methods based on RSCU that incorporate the idea of a reference set will be better suited to measure bias strength.

CAI offers a measure that is usefully normalized and based on highly-biased genes, but the genes in question are only those that are highly expressed, excluding the possibility of any bias other than translational.

Scaled chi-squared is based on the probability that local codon usage differs from the genome's average, and will not allow comparisons among organisms with different average codon usage.

The effective number of codons method provides a nicely intuitive idea of bias strength; a stronger bias will clearly have a smaller number of effective codons. However, in its native form this method only provides values for genes rather than a value based on the dominant bias. ICDI has similar drawbacks to those of Nc and also lacks the intuition that makes Nc so attractive.

SCCI does an iterative search for the genes with the most “strongly self-consistent” bias. This method is said to find the strongest, dominant bias in a genome, but the authors’ use of “strength” is somewhat different than the definition used here. A gene’s SCCI value, which is indicative of the gene’s degree of bias, is a measure of the degree of majority of the codons therein rather than a measure of the degree to which those codons are biased. That is, SCCI depends on the *number* of major codons rather than those codons’ usage relative to their synonyms, and thus matches the idea of adherence more closely than that of strength. Larger weights correspond to more highly-preferred codons, so a genome with higher SCCI scores on average may be more strongly biased than a genome with lower SCCI scores. But, it may also be possible for a gene with a mix of extremely high and extremely low weights to have a very similar SCCI to a gene with a more median and homogenous set of weights, which should not be the case when measuring the strength of the bias.

A strength criterion was introduced in (Carbone, Kepes et al. 2005) and used in conjunction with a ribosomal criterion to measure the tendency of a genome towards translational bias. The criterion was calculated as the *L2* distance between the genome’s dominant bias and average usage (see Section 3.2.3); if a genome had a high strength criterion and was characterized by translational bias, it was said to be strongly characterized. Like the adherence measure discussed above, the strength criterion meets most of the requirements for a measurement of bias strength, but it is unclear whether it effectively captures the desired properties of bias strength. If the average usage is also very biased, this number will be small even for a very strongly-biased genome.

Of all the techniques discussed herein, the best choices for a bias strength metric are SCCI and the strength criterion derived from SCCI. Neither of these methods exactly matches the criteria set forth for a bias strength metric; SCCI appears to be more closely related to a genome's adherence to a bias, and the strength criterion measures differences between the dominant bias and average codon usage rather than properties of the dominant bias alone. However, the precise biological implications of a bias' strength, adherence, and difference from average usage have not yet been studied. As they are somewhat conceptually similar, it is possible that they are closely related to the point of being interchangeable. The remainder of this chapter examines these two metrics along with a proposed metric for bias strength to evaluate their degree of similarity and determine which is best suited to compute bias strength.

4.2.4. Calculation of metrics

4.2.4.1. Adherence

The adherence metric is based on the results of the SCCI method used in the previous chapter and can be calculated using the codon weight vector and reference set of genes output by SCCI. Each gene receives an index computed by taking the geometric mean of the weights belonging to the codons in the gene sequence. A large score (approaching

1.0) indicates a gene that adheres strongly to the bias defined by the codon weight vector by containing a large number of codons with high weights; a small score corresponds to weak adherence and indicates a gene that uses major codons infrequently, if at all. Because the aim is to measure properties of the dominant bias, a genome's adherence score will be calculated by taking the mean of the SCCI values for only those genes included in the reference set rather than the values for the entire genome.

4.2.4.2. Distance

The strength criterion is also based on the results of SCCI; it is the ½- *l1* distance between the codon weight vector for the dominant bias and the codon weight vector that corresponds to the global bias.

$$Dist = \frac{\sum_{i=1}^{59} \left| w_i^g - w_i^d \right|}{2} = \frac{1}{2} \, l_1 \left(w^g \; , \; w^d \right) \tag{18}$$

Note that this equation also utilizes only the 59 codons able to display bias, as discussed previously. Conceptually, the strength criterion indicates the number of major codons that change between the average and the dominant bias and will thus be referred to as a distance metric to avoid confusing it with the metric for true bias strength. Although this measure has been previously applied to the idea of bias strength, its use as a stand-alone technique is dubious because it uses the genome's global bias as a baseline rather than some value that can be applied to all genomes. Another possible issue is that it may allow synonymous groups containing different numbers of codons to contribute differently to the final measure.

4.2.5. Proposed bias strength metric

The goal of this metric is to provide some measure of the extent to which codon usage in a biased genome differs from equal usage of synonymous codons. The input to the metric consists of the results obtained by applying the SCCI algorithm to a genome; this includes the codon weight vector and reference set of genes (including sequences) described in Section 3.2.3. Computation will be a two-step process; first the contribution of each group of synonyms is computed, then these values are averaged to obtain the final result of the metric.

To obtain the contribution value from each group of synonyms, first the number of times each codon in the group appears in the reference set is counted. Each occurrence of a codon *c* is assigned a weight computed as follows:

$$occ_c = \frac{1}{n} \Big/ x_c \tag{19}$$

Where *n* is the number of synonyms and *x* is the number of occurrences of codon *c*. All occurrences of a codon thus have equal weights, and the weights are inversely proportional to the frequency of the codon. The contribution value *V* for synonymous group *A* is then computed by ordering the occurrence weights by codon majority, summing the top half of the weights, and then multiplying the sum by two. The sum of all occurrence weights is always equal to *n*, so this step is necessary in order to represent the distribution of occurrences among the possible codons as a percentage. Choosing a different cutoff point has little effect on the resulting value unless a cutoff point within ~2% of 0% or 100% is used. Fifty percent was chosen because it simplifies the normalization step necessary to convert the value to a percentage. Because the occurrence weights are inversely proportional to codon frequency, and the number of occurrences included in the sum is fixed, smaller occurrence weights (which correspond to more frequently-used codons) result in a smaller contribution value for the group. A group whose codons appear with equal frequency has a contribution value approaching 1.0, and an extremely biased group has a lower bound equal to 1/*n*.

$$V_A = 2 \sum_{c=0}^{\frac{count(A)}{2}} occ_c \qquad (20)$$

Where V is the contribution value for synonymous group A and count(A) refers to the total number of occurrences of the codons included in group A. There are eighteen synonymous groups capable of biased usage, so the final value of the strength metric is calculated by taking the average of the group contributions as shown below.

$$BS_{total} = \frac{\sum_{A} V_A}{18} \qquad (21)$$

The value calculated by this process is proportional to the percentage of available codons that are utilized by the genome. If equal usage is assumed, this value is 1.0; the genome uses all the available codons. As a genome becomes more biased in its usage, the value decreases; the genome uses proportionally fewer codons because the codons that appear do so frequently.

Calculating bias strength in this way has the benefit of allowing the contribution of each amino acid to scale with the number of synonymous codons it possesses with no additional normalization. Consider the case of two amino acids, one with six codons and one with four. If both of these amino acids use only one codon, then clearly they both display the maximum amount of bias possible. But, the major codon in the six-codon group is preferred over five synonyms, while the other major codon is preferred over only three. Even though both groups are as biased as they can be, the six-codon group is more biased because it excludes more alternative codons from use. Applying the proposed bias strength metric to these synonymous groups gives values of 0.17 for the six-codon group and 0.25 for the four-codon group. A two-codon group is capable of displaying a minimum value of 0.5. The overall minimum value for the metric is therefore 0.37, which is calculated using the minimum values for each amino acid. This value corresponds to a genome in which one codon is used per amino acid, to the exclusion of all other codons.

$$SM_{\min} = \frac{9\left(\frac{1}{2}\right) + \left(\frac{1}{3}\right) + 5\left(\frac{1}{4}\right) + 3\left(\frac{1}{6}\right)}{18} = 0.37 \tag{22}$$

One possibility for which this metric does not account is the fact that if all the occurrences of two or more codons fall above the 50% split, the usage of these codons relative to each other no longer matters. The desired outcome of this situation is less clear than in the examples presented above; no biological insight into the “correct” answer is available. Because the correct approach is not obvious, the fact that the metric does not differentiate here is less of a problem than it would be otherwise.

4.2.6. Evaluation of metrics

The ten most strongly-biased and ten most weakly-biased organisms are selected by visual inspection of their codon weight vectors according to the criteria used in defining bias strength; genomes with a consistently wide gap between the weights of major and non-major codons are considered strongly biased and genomes with very homogenous weights are considered weakly biased. Only twenty genomes total are ranked in this way because only the genomes at the extreme ends of the spectrum are easily distinguished by visual means. Although this means of evaluation is somewhat subjective, there are no other means of comparison than the metrics under study.

The three metrics described above are applied to all forty organisms, generating a separate ranking and set of values for each metric. The top and bottom ten from each are selected to determine how many organisms they share with the results of the visual classification. Results of this evaluation are shown in Table 4.

Table 4. Metric evaluation

Most biased

Visual	*Adherence*		*Strength Metric*		*Distance*	
Wigglesworthia glossinidia	Mesoplasma florum	0.86	Saccharopolyspora erythraea	0.42	Bacillus subtilis	11.93
Pseudomonas aeruginosa	Saccharopolyspora erythraea	0.83	Thermus thermophilus	0.42	Bdellovibrio bacteriovorus	9.98
Xanthomonas oryzae	Wigglesworthia glossinidia	0.83	Sorangium cellulosum	0.43	Lactobacillus plantarum	9.62
Saccharopolyspora erythraea	Sorangium cellulosum	0.82	Mycoplasma pulmonis	0.43	Yersinia pestis Angola	9.14
Sorangium cellulosum	Staphylococcus aureus	0.81	Ralstonia solanacearum	0.44	Salmonella enterica	8.99
Blochmannia floridanus	Thermus thermophilus	0.81	Mycobacterium smegmatis	0.44	Lactococcus lactis	8.91
Onion yellows phytoplasma	Ralstonia solanacearum	0.80	Pseudomonas aeruginosa	0.44	Polynucleobacter necessarius	7.93
Thermus thermophilus	Listeria innocua	0.80	Wigglesworthia glossinidia	0.44	Haemophilus influenzae	7.88
Lactococcus lactis	Onion yellows phytoplasma	0.79	Onion yellows phytoplasma	0.44	Anaplasma marginale	7.40
Clostridium perfringens	Clostridium perfringens	0.79	Blochmannia floridanus	0.45	Listeria innocua	6.85
	5 of 10		7 of 10		1 of 10	

Least biased

Visual	*Adherence*		*Strength metric*		*Distance*	
Anaplasma marginale	Methylobacillus flagellatus	0.68	Anaplasma marginale	0.64	Thermus thermophilus	1.55
Polynucleobacter necessarius	Prochlorococcus marinus	0.70	Yersinia pestis Angola	0.60	Sorangium cellulosum	1.63
Yersinia pestis Angola	Chlamydia trachomatis	0.70	Polynucleobacter necessarius	0.59	Saccharopolyspora erythraea	1.68
Lactobacillus plantarum	Anaplasma marginale	0.70	Listeria innocua	0.58	Pseudomonas aeruginosa	1.76

Listeria innocua	Bartonella bacilliformis	0.71	Chlamydia trachomatis	0.57	Wigglesworthia glossinidia	1.99
Bacillus subtilis	Synechococcus sp. WH 8102	0.72	Bacillus subtilis	0.56	Buchnera aphidicola	2.08
Wolbachia endosymbiont	Bdellovibrio bacteriovorus	0.72	Acholeplasma laidlawii	0.56	Ralstonia solanacearum	2.16
Nanoarchaeum equitans	Lactobacillus plantarum	0.72	Lactobacillus plantarum	0.55	Mycobacterium smegmatis	2.29
Chlamydia trachomatis	Yersinia pestis Angola	0.72	Nanoarchaeum equitans	0.54	Lawsonia intracellularis	2.31
Prochlorococcus marinus	Bacillus subtilis	0.73	Staphylococcus aureus	0.54	Clostridium perfringens	2.32
	6 of 10		8 of 10		0 of 10	

The extent to which the three metrics are correlated with each other is determined by means of Pearson's correlation coefficients. This is done to determine how much unique information is captured by each metric, given that the metrics are based on very similar data.

Table 5. Pearson's correlation coefficients among metrics

	Adherence	Distance
Bias strength	-0.54	0.66
Adherence		-0.44

The biggest open question in the results of the previous chapter is the biological meaning of the second principal component of the organisms in codon usage space. Pearson's correlation coefficients are again used to determine whether any of the metrics are related to the second principal component.

Table 6. Pearson's correlation coefficients between metrics and second PC

	PC2
Bias strength	-0.50
Adherence	0.14
Distance	-0.85

Evidence in the literature, as well as the previous chapter, indicates that GC content has a major effect on the codon usage of an organism. As an example of a possible use of the proposed bias strength metric, its relationship to GC content is explored by plotting each organism's bias strength as a function of its percent GC-content.

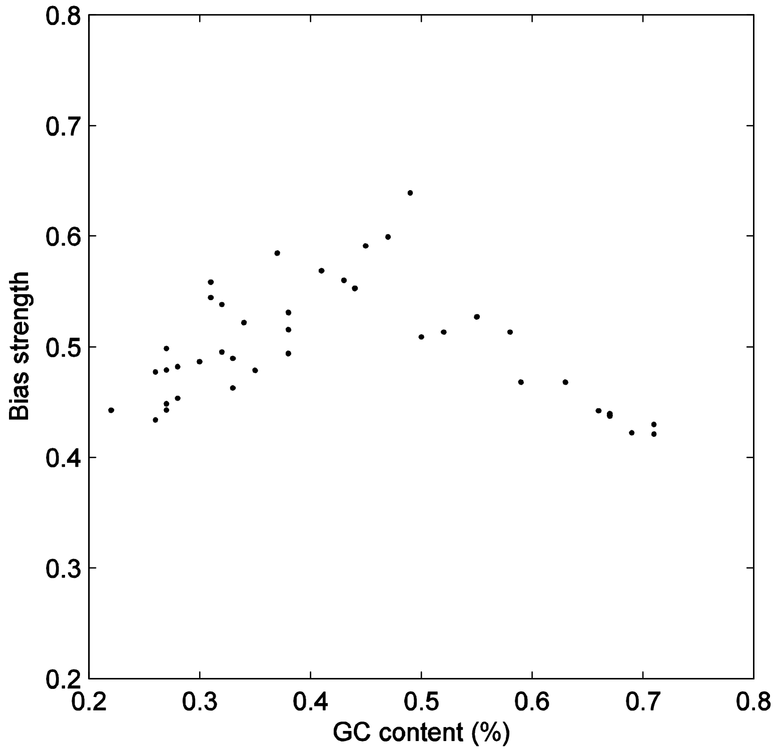

Figure 7. Bias strength as a function of GC content

4.3. Results

The ten organisms with the strongest bias as selected by the bias strength metric contain six of the ten selected by visual inspection; nine of the ten most weakly biased organisms match between those chosen by visual inspection and those ranked by the strength metric. Only five of the ten strongest and six of the ten weakest match as ranked by the adherence measure. The performance of the distance measure with regards to measuring bias strength is abysmal; the ten organisms with the largest distance measure contain more of the weakly-biased organisms than strongly-biased, with similar results for the ten organisms with the smallest distance. Of these three measures, the proposed strength metric does the best job of determining the strength of a bias according to the criteria set forth in Section 4.2.1.

The correlation coefficients among the three metrics indicate that some degree of correlation exists among all three. Adherence has a moderate negative correlation with distance, while bias strength has a strong negative correlation with adherence and a strong positive correlation with distance. In general, this shows that a strongly-biased genome has little disparity between its dominant bias and its average codon usage, and its genes have a high degree of adherence to the dominant bias. Weakly-biased genomes tend to have greater disparity between dominant bias and average usage, and lower adherence to the dominant bias. The trend in adherence values is consistent with the expectation that a strongly-biased genome will also have a high degree of adherence to its bias, but the trend of the distance measure directly refutes the prior hypothesis that a large distance measure is indicative of a strong bias. While this relationship may hold true when considering translational bias alone, it clearly does not generalize to all types of bias.

Because of the high degree of correlation among the three metrics under evaluation, it is unlikely that applying all three to the study of a group of genomes will supply significantly more information than if only one of the three is used. The metrics instead show a single trend that appears to hold true for biased codon usage in general, which may be stated as follows. If a genome is strongly biased, the bias is global to the genome, while a weaker bias may be limited to a subset of genes.

Table 6 shows that, of the three metrics, the distance measure is the most closely correlated to the second principal component of codon usage space with a very strong negative correlation. Bias strength has a moderate negative correlation and adherence has very little. It is therefore likely that the distance measure is the unknown property that accounts for the distribution of the forty organisms under study along the second principal component, making GC content and the distance between dominant and average codon usage the primary factors that account for the separation between free-living and obligate intracellular prokaryotes in codon usage space. Other analyses using codon usage space also found that GC content was responsible for the first principal component, while the second principal component was a property unique to the groups of organisms being analyzed. For example, optimal growth temperature was highly correlated with the second principal component in a comparison between mesophiles and thermophiles (Carbone, Kepes et al. 2005).

It is thus possible that the difference between dominant and average codon usage in a genome is related to some biological quality that distinguishes host-dependent, intracellular organisms from those that live on their own. Because the distance measure was originally used as a strength criterion for translational codon usage bias, these results suggest but do not prove that some free-living prokaryotes are characterized by a stronger translational bias than other free-livers and most obligate intracellular prokaryotes. However, it has already been stated that the strength criterion was not sufficient to determine whether a genome is characterized by translational bias at all, so further investigation is necessary to determine the validity of this theory.

In order to determine whether the concept of bias strength as defined here can provide any information with regards to the strength of known types of codon usage bias, the relationship between bias strength and GC content was examined. GC content was chosen for this comparison because it is known to be one of the most influential effects on the codon usage of a genome. Figure 7 shows a clear non-linear relationship between the bias strength and GC content of the forty organisms previously examined in this thesis. Genomes with a strong GC-content bias, i.e. those with a relatively high GC- or AT-content, are strongly biased, while genomes with moderate GC content (around 50%) are weakly biased. Because GC content is known to be global to the genome, these results serve to validate the general trend observed above, which suggests that strong biases are also global to the genome. These results also suggest that the bias strength metric proposed herein may be a useful tool for determining the relative strengths of different types of codon usage bias, in addition to GC-content bias.

5. Conclusions and future work

5.1. Contribution

The research described in Chapter 3 applies existing methodology for comparing biologically different groups of organisms in codon usage space to biological groups that had not previously been examined in this way. The results of Chapter 3 show that the concept of codon usage space as an indication of biological similarity remains valid for a comparison of free-living to obligate intracellular prokaryotes, thus further validating the use of codon usage space as a comparison tool. A great deal of the discrimination between these two groups of organisms is provided by their GC content. This is consistent with the results of previous research in this area. A large portion of the remaining discrimination is provided by an unknown factor, which is later identified in Chapter 4.

The existing literature on codon usage bias contains several mentions of the concept of bias strength, but provides no clear definition of the term. The fourth chapter of this thesis clarifies this, as well as other terminology associated with properties of codon usage bias, providing a concrete definition of bias strength, bias adherence, and a distance measure that indicates the variation between codon usage in the dominant bias and the genome's average codon usage. A metric intended to measure the bias strength of a genome according to this definition is proposed. This metric is demonstrated to capture the characteristics delineated in the given definition better than any existing measure of bias strength. A comparison of the codon usage bias metrics examined in Chapter Four suggest the existence of a general trend that may apply to biased codon usage in general, which may be stated as follows: a strong dominant bias is global to the genome and has high adherence, while a weak dominant bias does not affect the entire genome and has lower adherence.

In Chapter Three, an unknown factor was responsible for the separation between some of the organisms along the second principal component. An evaluation of the metrics examined in Chapter Four indicates that the distance measure is very strongly correlated with the organisms' position along the second principal component, suggesting that this property of the organisms' genomes is responsible for their separation. The relationship between the GC content of a genome and the genome's bias strength is also examined, and shows that genomes with extreme GC content are also strongly biased while genomes with moderate GC content are more weakly biased. This suggests that GC content bias is a stronger form of codon usage bias than the bias caused by other factors, and also indicates that the proposed bias strength metric may be useful for evaluating the relative strengths of different types of codon usage bias.

5.2. Future work

The results of the research described in this thesis offer several opportunities for further work on this topic. Chapter 3 suggests that the biological difference between free-living and obligate intracellular prokaryotes may be related in part to the tendencies of these types of organisms towards translational bias, but this possibility is not proven. It would be interesting to determine whether translational bias is responsible for this biological difference, or if it is not responsible, to find the biological characteristic that is responsible.

Chapter 4 suggests the existence of a trend that may apply to all types of codon usage bias. The application of the methodology used to deduce this trend could be applied to a data set containing a larger number of organisms in order to prove or disprove its existence. Chapter 4 also suggests that the bias strength metric proposed in this thesis may be a useful tool for determining the relative strength of different types of codon usage bias. An interesting exercise would be to apply the bias strength metric to new organisms, grouped by the types of codon usage bias their genomes display, in order to determine whether bias strength has any correlation with the tendency towards other types of bias than GC-content. If the hypothesis that strong biases are also global to the genome is confirmed, this exercise could provide useful information with regards to the characteristics of other types of codon usage bias, which may make it easier to identify them in the future.

Appendix A. Ruby source code

This appendix contains the source code used to implement the SCCI algorithm and calculate the bias strength and adherence metrics. These tasks were implemented in the Ruby programming language.

A.1. Utility.rb

The Utility.rb file contains custom extensions to build-in Ruby classes, as well as helper methods and data structures related to the general processing of the genetic code rather than any specific task.

```
#######################################
# Extensions to built-in Array class   #
#######################################
class Array

  # Accumulates the results of performing 'yield' on each element, n
holds result.
  #  Ex, if n = 0 and yield contains n + value, inject will compute the
sum of the array.
      def inject(n)
            each{ |value| n = yield(n, value) }
            n
      end # end inject


  # sum the numbers in the array
      def sum
            inject(0) { |n, value| n + value}
      end # end sum


  # Calculates the standard deviation of the numbers in an array
  def stdev(avg)
    stdev = self.inject(0){
      |n, value|
      n + (value.to_f - avg.to_f) ** 2
    }
    ( stdev / self.size ) ** 1.0/2.0
  end # end stdev


  # Given a parameter array (assumed to be the same length as self),
returns a new
  #  array of the same length.  The ith element of the new array is
computed by
  #  performing yield on the ith elements of self and the parameter
array.
  #  If the parameter is not the same length as self, fill the new
array with nil.
      def pairwise(arr2)
            newarr = Array.new(self.length, nil)

            if self.length == arr2.length
                  for index in 0 ... self.length
                        newarr[index] = yield(self[index], arr2[index])
                  end # end for
            end # end if

            newarr
      end # end pairwise

end # end Array



#######################################
# Extensions to built-in String class #
#######################################
class String

      # Similar to String#each, but iterates through the string in 3-
character codons
  # If self.length is not a multiple of 3, the odd characters at the
end will be ignored.
  # This should be modified by adding a parameter to select the length
of substring used
  #  to iterate, instead of forcing a 3-character length.
      def each_codon
            position = 0

    # This check prevents out-of-bounds errors if self.length is not
    #  a multiple of 3
            while position <= self.length-3
                  yield self[position, 3]
                  position = position + 3
            end # end while

      end # end each_codon

  def each_window(winSize)
    position = 0

    while position <= self.length-winSize
```

```
      yield self[position, winSize]
      position = position + winSize
    end # end while

    if position < self.length
      yield self[position, self.length-position]
    end # end if
  end # end each_window


  # Given a string fragment, returns self with all instances of that
fragment removed.
  # Intended for use with 3-character codons but works with a parameter
of any length -
  #  if self is shorter than frag, or if self.length is not a multiple
of frag.length, the extra
  #  characters will be unmodified.
      def delete_codon(frag)
            position = 0
            modSelf = self
    fragLen = frag.length

    # Prevents out-of-bounds errors
            while position <= modSelf.length-fragLen

                  if modSelf[position,fragLen] == frag
                        modSelf = modSelf.slice(position,fragLen)
                  else
                        position = position + fragLen
                  end # end if

            end # end while

            modSelf
      end # end delete_codon

end # end String

#######################################
# Extensions to built-in Float class  #
#######################################
class Float

  # Sets the precision of self to digits by converting to a string,
rounding the
  #  string to digits characters, then converting back to a float.
  def setPrecision(digits)
    ("%.#{ digits }f"%self).to_f
  end # end setPrecision

end # end Float

#######################################
# Extensions to built-in File class   #
#######################################
class File
```

```
      # Files with gene info contain 5 lines of information per gene -
this allows all the info for
      #  one gene to be read at once
      def each_gene
            while !self.eof
                  gname = self.gets.to_s.chomp.delete("<").delete(">")
                  gseq = self.gets.to_s.chomp
      gscci = self.gets.to_s.chomp
                  gstr = self.gets.to_s.chomp
                  gcounts = self.gets.to_s.chomp

                  yield(gname,gseqi, gscci, gstr,gcounts)
            end # end while
      end # end each_gene
end # end File


module Utility

  attr :codonSet

  GCN_CODONS = 0
  ALL_CODONS = 1


  def setCodonSet(set)
    if set != GCN_CODONS && set != ALL_CODONS
      @codonSet = nil
    else
      @codonSet = set
    end
  end # end setCodonSet


  # Number of GC-neutral families
      # GC_FAM_COUNT = 17
  def getFamCount
    if @codonSet == GCN_CODONS
      return 17
    elsif @codonSet == ALL_CODONS
      return 20
    else
      puts "Error: codonSet not initialized"
      return -1
    end
  end # end getFamCount


  # Number of codons in the GC-neutral families
  def getCodonCount
    if @codonSet == GCN_CODONS
      return 38
    elsif @codonSet == ALL_CODONS
      return 59
    else
      puts "Error: codonSet not initialized"
      return -1
    end
  end # end getCodonCount


  # Flags for loading coefficients from file
  CODON_COEFF = 0
  FAM_COEFF = 1
  ALL_COEFF = 2


      # Array of GC-neutral family names
  def getFamList
    if codonSet == GCN_CODONS
      return [
        'gly1',
        'gly2',
        'ala1',
        'ala2',
        'val1',
        'val2',
        'leu1',
        'leu2',
        'ile1',
        'pro1',
        'pro2',
        'ser1',
        'ser2',
        'thr1',
        'thr2',
        'arg1',
        'arg2'
      ]
    elsif codonSet == ALL_CODONS
      return [
        'gly',
        'ala',
        'val',
        'pro',
        'leu',
        'ile',
        'ser',
        'thr',
        'arg',
        'asp',
        'glu',
        'trp',
        'met',
        'lys',
        'cys',
        'his',
        'tyr',
        'phe',
        'asn',
        'gln',
      ]
    else
      puts "Error: codonSet not initialized"
      return nil
    end
  end # end getFamList


  # Array of GC-neutral family names
  # Static version
  def Utility.getFamList
    if codonSet == GCN_CODONS
      return [
        'gly1',
        'gly2',
        'ala1',
        'ala2',
        'val1',
        'val2',
        'leu1',
        'leu2',
        'ile1',
        'pro1',
        'pro2',
        'ser1',
        'ser2',
        'thr1',
        'thr2',
        'arg1',
        'arg2'
      ]
    elsif codonSet == ALL_CODONS
      return [
        'gly',
        'ala',
        'val',
        'pro',
        'leu',
        'ile',
        'ser',
        'thr',
        'arg',
        'asp',
        'glu',
        'trp',
        'met',
        'lys',
        'cys',
        'his',
        'tyr',
        'phe',
        'asn',
        'gln',
      ]
    else
      puts "Error: codonSet not initialized"
      return nil
    end
  end # end getFamList
```

```
def codonIndex(input)
  #puts input
  if input != 0 && input.to_i == 0   # input is a string (codon)
    if @codonSet == GCN_CODONS
      if GCN_CODON_LIST.include? input
        return CODON_TO_INDEX[input]
      else
        return nil
      end
    elsif @codonSet == ALL_CODONS
      return CODON_TO_INDEX[input]
    else # error
      puts "Error: codonSet not initialized"
      return nil
    end
  else  # input is an integer (index)
    if @codonSet == GCN_CODONS
      if GCN_CODON_LIST.include? INDEX_TO_CODON[input.to_i]
        return INDEX_TO_CODON[input.to_i]
      else
        return nil
      end
    elsif @codonSet == ALL_CODONS
      return INDEX_TO_CODON[input.to_i]
    else # error
      puts "Error: codonSet not initialized"
      return nil
    end
  end
end # end codonIndex


def codonToGroup(codonIndex)
  if @codonSet == GCN_CODONS
    return CODON_TO_GC_FAMILY[codonIndex.to_i]
  elsif @codonSet == ALL_CODONS
    return CODON_TO_AA[codonIndex.to_i]
  else
    puts "Error: codonSet not initialized"
    return nil
  end
end # end codonGroup


def groupToCodons(groupIndex)
  if @codonSet == GCN_CODONS
    return GC_FAMILY_TO_CODONS[groupIndex.to_i]
  elsif @codonSet == ALL_CODONS
    return AA_TO_CODONS[groupIndex.to_i]
  else
    puts "Error: codonSet not initialized"
    return nil
  end
end # end groupToCodon
```

```
def Utility.groupToCodons(groupIndex)
  if @codonSet == GCN_CODONS
    return GC_FAMILY_TO_CODONS[groupIndex.to_i]
  elsif @codonSet == ALL_CODONS
    return AA_TO_CODONS[groupIndex.to_i]
  else
    puts "Error: codonSet not initialized"
    return nil
  end
end # end groupToCodon


# AA to codon list
AA_TO_CODONS = {
  0 => [43, 41, 40, 42],
  1 => [39, 37, 36, 38],
  2 => [47, 45, 44, 46],
  3 => [23, 21, 20, 22],
  4 => [60, 62, 31, 29, 28, 30],
  5 => [15, 13, 12],
  6 => [55, 53, 52, 54, 11, 9],
  7 => [7, 5, 4, 6],
  8 => [27, 25, 24, 26, 8, 10],
  9 => [35, 33],
  10 => [32, 34],
  13 => [0, 2],
  14 => [59, 57],
  15 => [19, 17],
  16 => [51, 49],
  17 => [63, 61],
  18 => [3, 1],
  19 => [16, 18]
}


# codon to AA index
CODON_TO_AA = {
  0 => 13, 1 => 18, 2 => 13, 3 => 18,
  4 => 7, 5 => 7, 6 => 7, 7 => 7,
  8 => 8, 9 => 6, 10 => 8, 11 => 6,
  12 => 5, 13 => 5,           15 => 5,
  16 => 19, 17 => 15, 18 => 19, 19 => 15,
  20 => 3, 21 => 3, 22 => 3, 23 => 3,
  24 => 8, 25 => 8, 26 => 8, 27 => 8,
  28 => 4, 29 => 4, 30 => 4, 31 => 4,
  32 => 10, 33 => 9, 34 => 10, 35 => 9,
  36 => 1, 37 => 1, 38 => 1, 39 => 1,
  40 => 0, 41 => 0, 42 => 0, 43 => 0,
  44 => 2, 45 => 2, 46 => 2, 47 => 2,
  49 => 16, 51 => 16,
  52 => 6, 53 => 6, 54 => 6, 55 => 6,
  57 => 14, 59 => 14,
  60 => 4, 61 => 17, 62 => 4, 63 => 17
}


# hash for index-to-codon, default nil for indices not in range
```

```
INDEX_TO_CODON = {
  0 => 'aaa', 1 => 'aac', 2 => 'aag', 3 => 'aat',
  4 => 'aca', 5 => 'acc', 6 => 'acg', 7 => 'act',
  8 => 'aga', 9 => 'agc', 10 => 'agg', 11 => 'agt',
  12 => 'ata', 13 => 'atc', 14 => 'atg', 15 => 'att',
  16 => 'caa', 17 => 'cac', 18 => 'cag', 19 => 'cat',
  20 => 'cca', 21 => 'ccc', 22 => 'ccg', 23 => 'cct',
  24 => 'cga', 25 => 'cgc', 26 => 'cgg', 27 => 'cgt',
  28 => 'cta', 29 => 'ctc', 30 => 'ctg', 31 => 'ctt',
  32 => 'gaa', 33 => 'gac', 34 => 'gag', 35 => 'gat',
  36 => 'gca', 37 => 'gcc', 38 => 'gcg', 39 => 'gct',
  40 => 'gga', 41 => 'ggc', 42 => 'ggg', 43 => 'ggt',
  44 => 'gta', 45 => 'gtc', 46 => 'gtg', 47 => 'gtt',
  48 => 'taa', 49 => 'tac', 50 => 'tag', 51 => 'tat',
  52 => 'tca', 53 => 'tcc', 54 => 'tcg', 55 => 'tct',
  56 => 'tga', 57 => 'tgc', 58 => 'tgg', 59 => 'tgt',
  60 => 'tta', 61 => 'ttc', 62 => 'ttg', 63 => 'ttt'
} # end INDEX_TO_CODON


CODON_TO_INDEX = {
  'aaa' => 0, 'aac' => 1, 'aag' => 2, 'aat' => 3,
  'aca' => 4, 'acc' => 5, 'acg' => 6, 'act' => 7,
  'aga' => 8, 'agc' => 9, 'agg' => 10, 'agt' => 11,
  'ata' => 12, 'atc' => 13, 'atg' => 14, 'att' => 15,
  'caa' => 16, 'cac' => 17, 'cag' => 18, 'cat' => 19,
  'cca' => 20, 'ccc' => 21, 'ccg' => 22, 'cct' => 23,
  'cga' => 24, 'cgc' => 25, 'cgg' => 26, 'cgt' => 27,
  'cta' => 28, 'ctc' => 29, 'ctg' => 30, 'ctt' => 31,
  'gaa' => 32, 'gac' => 33, 'gag' => 34, 'gat' => 35,
  'gca' => 36, 'gcc' => 37, 'gcg' => 38, 'gct' => 39,
  'gga' => 40, 'ggc' => 41, 'ggg' => 42, 'ggt' => 43,
  'gta' => 44, 'gtc' => 45, 'gtg' => 46, 'gtt' => 47,
  'taa' => 48, 'tac' => 49, 'tag' => 50, 'tat' => 51,
  'tca' => 52, 'tcc' => 53, 'tcg' => 54, 'tct' => 55,
  'tga' => 56, 'tgc' => 57, 'tgg' => 58, 'tgt' => 59,
  'tta' => 60, 'ttc' => 61, 'ttg' => 62, 'ttt' => 63
} # end CODON_TO_INDEX


# family to codon list
GC_FAMILY_TO_CODONS = {
          0 => [42, 41],
          1 => [40, 43],
          2 => [38, 37],
          3 => [36, 39],
          4 => [46, 45],
          5 => [44, 47],
          6 => [30, 29],
          7 => [28, 31, 62],
          8 => [12, 15],
          9 => [22, 21],
          10 => [20, 23],
          11 => [54, 53, 9],
          12 => [52, 55, 11],
          13 => [6, 5],
          14 => [4, 7],
          15 => [26, 25],
          16 => [24, 27, 10]
    } # end FAMILY_TO_CODONS


  # codon to family
  CODON_TO_GC_FAMILY = {
    4 => 14, 5 => 13, 6 => 13, 7 => 14,
    9 => 11, 11 => 12, 12 => 8, 15 => 8,
    20 => 10, 21 => 9, 22 => 9, 23 => 10,
    24 => 16, 25 => 15, 26 => 15, 27 => 16,
    28 => 7, 29 => 6, 30 => 6, 31 => 7,
    36 => 3, 37 => 2, 38 => 2, 39 => 3,
    40 => 1, 41 => 0, 42 => 0, 43 => 1,
    44 => 5, 45 => 4, 46 => 4, 47 => 5,
    52 => 12, 53 => 11, 54 => 11, 55 => 12,
    62 => 7, 10 => 16
  }


  GCN_CODON_LIST = [
    'ggg', 'ggc', 'gga', 'ggt',
    'gcg', 'gcc', 'gca', 'gct',
    'gtg', 'gtc', 'gta', 'gtt',
    'ctg', 'ctc', 'cta', 'ctt',
    'ttg', 'ata', 'att', 'ccg',
    'ccc', 'cca', 'cct', 'tcg',
    'tcc', 'agc', 'tca', 'tct',
    'agt', 'acg', 'acc', 'aca',
    'act', 'cgg', 'cgc', 'cga',
    'cgt', 'agg'
  ]


  # Given a DNA sequence, return a vector indicating the number of
times each GCN-fam codon
  #  appears in the sequence
  def countCodons(sequence)
    counts = Array.new(64, 0)

    sequence.each_codon{
      |codon|
      cIndex = codonIndex(codon)
      counts[cIndex] += 1 if cIndex != nil
    } # end each_codon

    counts
  end # end Utility.countCodons
end # end Utility
```

A.2. Genome.rb

This file implements the Genome and Gene objects, which are used for data storage, organization, and processing such as counting and storing the codon counts for individual genes. The Genome object also contains the code necessary to compute the bias strength metric.

```
require "Utility.rb"

class Genome
  include Utility

  attr_reader :glist, :gmBias, :gmAdher, :numPhages
  attr_reader :refSet, :codonWeights, :famWeights, :codonRelUsage

  attr_writer :refSet, :codonWeights, :numPhages, :glist, :famWeights
  attr_writer :codonRelUsage


  def initialize(gb, ga, np, rs, cwv, fwv, ruv, gl)
    @gmBias = gb
    @gmAdher = ga
    @numPhages = np
    @refSet = rs
    @codonWeights = cwv
    @famWeights = fwv
    @codonRelUsage = ruv
    @glist = gl
  end


  def Genome.newEmpty
    Genome.new(-1, -1, -1, nil, nil, nil, nil, Array.new)
  end # end newEmpty


  # Loads a formatted genome file into a Genome object, returns the
object.
  def Genome.loadFile(fileName)
    gb = -1
    ga = -1
    np = -1
    rs = nil
    cwv = nil
    fwv = nil
    ruv = nil
    gl = Array.new

    if !FileTest.exists? fileName
      puts "Genome#loadFile: #{fileName} does not exist"
      return Genome.newEmpty
    end
    infile = File.new(fileName, "r")
    line = infile.gets.chomp
    value = 0.0

    if line == '/nphages'
      line = infile.gets.chomp
      value = line.to_i

      # If not an integer
      if value == 0 && line != "0"
        puts "Genome#loadFile: /nphages invalid in #{fileName}"
      else
        np = value
      end

      # If current line is not a flag, read next line
      if line[0] != '/'
        line = infile.gets.chomp
      end

    else # no /np
      puts "Genome#loadFile: /nphages not found in #{fileName}"
    end # end if /np

    if line == '/genes'
      if infile.eof
        puts "Error: empty genelist"
      else

        while !infile.eof
          gname = infile.gets.chomp
          gCC = infile.gets.chomp
          gl.push(Gene.new(gname, nil, -1, -1, gCC))
        end

      end
    else
      puts "Error: /genes not found"
    end # end if /gl

    Genome.new(gb, ga, np, rs, cwv, fwv, ruv, gl)
  end


  # Reads genes (name and sequence only) from a file into @glist
  def readGenes(filename)
    success = 1
    fIndex = 0
            fileLines = IO.readlines( filename )

    # Each gene is represented in the file by two lines: the gene name
and the sequence.  So, the array is
    #  read 2 lines at a time - currIndex is the name, currIndex+1 is
the sequence.
    if fileLines.size > 0

      for currIndex in 0 ... (fileLines.size/2)
```

```
        name = fileLines[fIndex].to_s.chomp.delete("<").delete(">")

        if fIndex+1 < fileLines.length
          seq = fileLines[fIndex+1].to_s.chomp
          @glist[currIndex] = Gene.new(name, seq, -1, -1, nil )
        else
          if name != "" || fileLines.size % 2 != 0
            puts "Error in Genome#readGenes: #{ filename } does not
have even lines"
            puts "Read #{ @glist.size } genes"
            break
          end
        end

        fIndex += 2
      end # end for

    else
      success = 0
    end

            # Trailing whitespace in the file may result in empty gene
objects - this gets rid of them
            @glist = @glist.delete_if { |elt| elt.name == "" }

    success
  end # end readGenes


  # Calculate relative usage weights for families, store in @famWeights
  def setFamWeights
    success = 1
    @famWeights = Array.new(getFamCount, 0)

    if @glist.size > 0
      for index in 0 ... 64
        #puts index
        if codonIndex(index) != nil
          famIndex = codonToGroup(index)

          if famIndex == nil
            next
          end

          @glist.each{
            |gene|
            if gene.codonCounts == nil
              gene.codonCounts = Utility.countCodons(gene.sequence)
            end
            @famWeights[famIndex] += gene.codonCounts[index].to_i
          }

          if success == 0
            break
          end
        end # end if
      end # end for
      if success == 1
        maxWeight = @famWeights.max.to_f
        @famWeights.each_index{
          |famIndex|
          @famWeights[famIndex] = @famWeights[famIndex].to_f /
maxWeight
        }
      end

    else
      success = 0
    end

    success
  end # end setFamWeights


  # Calculate relative usage weights for codons
  def setRelUsage
    success = 1
    @codonRelUsage = Array.new(64, 0)
    sequenceCount = Array.new(64, 0)

    if @glist.size > 0
      for index in 0 .. 63
        #puts inx
        if codonIndex(index) != nil
          @glist.each{
            |gene|
            if gene.codonCounts == nil
              gene.codonCounts = countCodons(gene.sequence)
            end

            #puts gene.codonCounts.join(",") if index == 0

            if gene.codonCounts[index].to_i > 0
              sequenceCount[index] += 1
            end
          } # end @glist iteration

          @codonRelUsage[index] = sequenceCount[index].to_f /
@glist.size.to_f
        end # end if
      end # end for
    else # @glist is empty
      success = 0
    end

    success
  end # end setRelUsage


  def getBiasStrength(range,perc)
    aaList = getFamList
    aaCount = Array.new(aaList.size,0)
    aaPerc = Array.new(aaList.size,0)
    aaVals = Array.new
    ret = Array.new
    #outfile = File.new("bias_strength.csv","w")

    for aaInx in 0 ... aaList.size
      cdns = groupToCodons(aaInx)
      next if cdns == nil
      cdnCounts = Array.new(cdns.size,0)

      cdns.each_index{
        |cInx|
        cdnCounts[cInx] = cdnCounts[cInx].to_i
        @glist.each_index{
          |geneInx|
          if range == "d" && @refSet[geneInx].to_i == 0
            next
          end
          @glist[geneInx].codonCounts =
countCodons(@glist[geneInx].sequence) if @glist[geneInx].codonCounts ==
nil
          cdnCounts[cInx] +=
@glist[geneInx].codonCounts[cdns[cInx]].to_i
        }
        #puts cdnCounts.join(" ")
      }

      cdnCounts.sort!
      cdnCounts.reverse!

      cdnOcc = Array.new(cdnCounts.sum,0)
      occInx = 0

      cdnCounts.each{
        |cCount|
          for i in occInx ... (occInx + cCount)
            cdnOcc[i] = (1/cdns.size.to_f) / cCount
          end
          occInx += cCount
      }

      topHalf = (cdnCounts.sum.to_f/2).floor
      aaVals.push(cdnOcc[0,topHalf].sum)

      #aaCount[aaInx] = runningCount
      #aaPerc[aaInx] = runningCount/cdns.size.to_f

#ret.push(String.new("#{aaList[aaInx]},#{aaCount[aaInx]},#{aaPerc[aaInx
]}"))
    end

    # We want to account for half of the total codon occurrences;
    #  under conditions of perfect balance this would take (20-2)/2
codons
    ret.push(aaVals.sum.to_f/9)
    #~ ret.push(aaPerc.sum/(aaPerc.size-2).to_f)
    #~ ret.push(aaCount.sum.to_f/(59*perc))
    #aaPerc.sum/(aaPerc.size-2).to_f
    #outfile.close
  end


  # Count and set the number of phage-related genes in the given .out
file
  def countPhages(filename)
    success = 1
    fileLines = IO.readlines(filename)
    fileString = fileLines.join

    phageCount = 0
    geneCount = 0
    re = /CDS.*?\/translation/m
    match = fileString[re]

    while match != nil
      fileString[re] = ""
      geneCount += 1

      if match =~ /phage/ || match =~ /virus/ || match =~ /viral/ ||
match =~ /transpos/
        phageCount += 1
      end

      match = fileString[re]
    end # end while

    @numPhages = phageCount
    success
  end # end countPhages

  def writeGenome(filename)
    outfile = File.new(filename, "w")

    if @gmBias != -1
      outfile.puts '/bs'
      outfile.puts @gmBias.setPrecision(2)
    end

    if @gmAdher != -1
      outfile.puts '/ba'
      outfile.puts @gmAdher.setPrecision(2)
    end

    if @numPhages != -1
      outfile.puts '/nphages'
      outfile.puts @numPhages
    end

    if @refSet != nil
      outfile.puts '/rs'
      outfile.puts @refSet.join(",")
    end

    if @codonWeights != nil
      outfile.puts '/cw'
      outfile.puts @codonWeights.join(",")
    end

    if @famWeights != nil
      outfile.puts '/fw'
      outfile.puts @famWeights.join(",")
    end

    if @codonRelUsage != nil
      outfile.puts '/cu'
      outfile.puts @codonRelUsage.join(",")
    end

    if @glist != nil
      outfile.puts '/genes'
      @glist.each{
        |gene|
        outfile.print gene.to_s
      }
    end
  end # end writeGenome

end # end Genome


##########################################################
# Gene class
# Holds information on one gene: name and sequence come from .out
files, codonCounts can be
#  calculated from the sequence, and SCCI and distance from dominant
bias can be calculated after
#  the dominant bias has been calculated on the genome to which the
gene belongs
##########################################################
class Gene
  include Utility

  attr_reader :name, :sequence, :codonCounts, :SCCI, :dist
  attr_writer :codonCounts, :SCCI, :dist

  # Assumes parameters have been validated by calling method (missing
values replaced by flags, etc.)
  def initialize(gname, gseq, gSCCI, gdist, gCC)
    @name = gname
    @sequence = gseq
    @SCCI = gSCCI.to_f
    @dist = gdist.to_f

    if gCC == nil
      if @sequence != nil
        setCodonSet ALL_CODONS

        # Disregard initiation and stop codons
        #if @sequence[0,3] == 'atg'
          @sequence = @sequence[3, @sequence.length-3]
        #end
        #if @sequence[@sequence.length-3, 3] == 'taa' ||
@sequence[@sequence.length-3, 3] == 'tag' || sequence[sequence.length-
3, 3] == 'tga'
          @sequence = @sequence[0, @sequence.length-3]
        #end

        @codonCounts = countCodons(@sequence)
      end
      #puts @codonCounts.join(",")
    else
      @codonCounts = gCC.split(",")
    end # end if

    @codonCounts.each_index{
      |index|
      @codonCounts[index] = @codonCounts[index].to_i
    }

  end # end initialize


  # Each data member occupies one line, so a gene's full set of info
takes up 5 lines of the file.
  def to_s
    ret = ""

    if @name != nil
      ret = ret + @name + "\n"
    end

    #~ if @sequence != nil
      #~ ret = ret + @sequence + "\n"
    #~ end

    if @SCCI != -1
      ret = ret + @SCCI + "\n"
    end

    if @dist != -1
      ret = ret + @dist + "\n"
    end

    if @codonCounts != nil
      ret = ret + @codonCounts.join(",") + "\n"
      #~ @codonCounts.each_index{
        #~ |index|
        #~ cIndex = codonIndex(index)
        #~ ret = ret + codonIndex(index) + ":" +
@codonCounts[index].to_s + "\n" if cIndex != nil
      #~ }
      #~ ret = ret + "\n"
    end

    #"#{ @name }\n#{ @sequence }\n#{ @SCCI }\n#{ @dist }\n#{
@codonCounts.join(",") }\n"
    ret
  end # end to_s
end # end Gene
```

A.3. Bias.rb

This file contains the implementation of the SCCI algorithm. Bias adherence is calculated using methods related to this algorithm.

```
require "Genome.rb"
require "Utility.rb"


class Bias
  include Utility

  attr :gn, :cs

  GCN_CODONS = 0
  ALL_CODONS = 1

  def initialize(dirname, cSet)
    @gn = Genome.loadFile("#{PROCESSED_DIR}/#{dirname}.dat")
    setCodonSet(cSet)
    @gn.setCodonSet(cSet)
    @cs = cSet

    if cSet == ALL_CODONS
      vectorFileName = "#{PROCESSED_DIR}/#{dirname}_all.dat"
    elsif cSet == GCN_CODONS
      vectorFileName = "#{PROCESSED_DIR}/#{dirname}_gcn.dat"
    else
      puts "blargh"
      vectorFileName = ""
    end

    vectorFile = File.new(vectorFileName, "r")
    line = vectorFile.gets.chomp

    if line == "/codonreluse"
      @gn.codonRelUsage = vectorFile.gets.chomp.split(",")
      line = vectorFile.gets.chomp
    else
      puts "Error in file format: #{vectorFileName}"
      return
    end

    if line == "/famreluse"
      @gn.famWeights = vectorFile.gets.chomp.split(",")
    else
      puts "Error in file format: #{vectorFileName}"
      return
    end
  end # end initialize

  def findDominantBias
    #@gn.setCodonSet(@cs)

    # Initialize reference set to include all genes
    refset = Array.new(@gn.glist.length, 1)

    # Set the final size of the ref set to 1% of the number of genes
    finalRefSize = (refset.length * 0.01).round

    # Current ref set size is total number of genes
    currRefSize = refset.length

    # Keep track of all previous refsets
    prevRefsets = Array.new

    # Initialize codon weight vector to zero
    cwv = Array.new(64, 0)

    # Import codon relative usage from the genome
    codonRelUsage = @gn.codonRelUsage

    # Determines whether oscillation between 2 reference sets has
occurred
    bounce = false
    increment = -1

    # flag to continue iteration
    continue = true

    counter = 0

    # "The algorithm is iterative..."
    while(continue)
      puts "\nIteration #{counter}"

      # Calculate cwv on current ref set
      puts "Calculating cwv"
      cwv = calcWeights(refset)

      # Calculate gSCCI for all genes using cwv
      puts "Calculating SCCI"
      @gn.glist.each{
        |gene|
        gene.SCCI = calcCountsSCCI(cwv, gene.codonCounts)
        #~ puts gene.name + "," + gene.SCCI.to_s
      }

      # Reset size of ref set to 1/2 of current size.  If the resulting
size < finalRefSize,
      #  set to finalRefSize.  If oscillation has occurred, decrement
size by one.
      if !bounce
        currRefSize = (currRefSize/2.0).round
        if currRefSize < finalRefSize
          currRefSize = finalRefSize
        end
      else # If refset is oscillating
        if refset != prevRefsets[prevRefsets.length-1] &&
prevRefsets.include?(refset)
          if currRefSize > 1
            puts "Decrementing currRefSize by one"
            currRefSize = currRefSize-1
          else
            puts "Refset not found, terminating iteration."
            refset = nil
            cwv = nil
            break
          end
        end
      end

      # Find new ref set: sort genes by gCAI, take top <currRefSize>
genes as ref set
      #prev2Refset = prevRefset
      #prevRefset = refset
      prevRefsets.push(refset)
      refset = findRefSet(currRefSize)

      #~ puts "Iteration " + counter.to_s + ": refsize = " +
currRefSize.to_s
      #~ refset.each_index{
        #~ |index|
        #~ puts @gn.glist[index].name if refset[index] == 1
      #~ }
      #~ puts
      #~ puts "Iteration " + counter.to_s + ": refsize = " +
currRefSize.to_s
      #~ cwv.each_index{
        #~ |cIndex|
        #~ puts codonIndex(cIndex) + "," + cwv[cIndex].to_s
      #~ }

      if refset == prevRefsets[prevRefsets.length-1] # refset is the
same 2 iterations in a row, we're done
        continue = false
      elsif prevRefsets.include? refset   # refset matches a previous
refset, oscillation has occurred
        puts "bouncing" if !bounce
        bounce = true
      end
      counter = counter+1
    end # end algorithm while

    @gn.refSet = refset
    @gn.codonWeights = calcWeights(refset)

    @gn.glist.each{
      |gene|
      gene.SCCI = calcCountsSCCI(cwv, gene.codonCounts)
    }
    @gn
  end # end findDominantBias

  def getBaselineWeights
    refset = Array.new(@gn.glist.size, 1)
    calcWeights(refset)
  end # end getBaselineWeights

  def getBaselineAdherence
    refset = Array.new(@gn.glist.size, 1)
    cwv = calcWeights(refset)
    bl = Array.new
    @gn.glist.each{
      |gene|
       bl.push(calcCountsSCCI(cwv, gene.codonCounts))
    }
    bl
  end # end getAdherence

  # Helper methods
  # Geometric mean of relUsage*weight for each codon in sequence
  def calcSeqGSCCI(cwv, sequence)
    numGCNCodons = 0
    scci = 1.0

    # count total number of GCN codons (L)
    sequence.each_codon{
      |codon|
      cIndex = codonIndex(codon)
      numGCNCodons += 1 if cIndex != nil
    } # end each_codon (counting L)

    # Calculate SCCI - 1/Lth power is taken as the product is
calculated so that it does not shrink to 0
    sequence.each_codon{
      |codon|
      cIndex = codonIndex(codon)
      scci = scci * ( cwv[cIndex].to_f ** (1.0/numGCNCodons) ) if
cIndex != nil
    } # end each_codon (calculating SCCI)
    scci
  end # end calcSeqGCAI

  def calcCountsGSCCI(cwv, counts)
    if cwv == nil || counts == nil
      return nil
    end

    numGCNCodons = 0
    scci = 1.0

    counts.each_index{
      |cIndex|
      if codonIndex(cIndex) != nil
        numGCNCodons += counts[cIndex]
      end
    }

    counts.each_index{
      |cIndex|
      if codonIndex(cIndex) != nil
        for i in 0 ... counts[cIndex]
          scci = scci * ( (cwv[cIndex] *
@gn.codonRelUsage[cIndex].to_f) ** (1.0/numGCNCodons) )
          #scci = scci * ( cwv[cIndex] ** (1.0/numGCNCodons) )
        end
      end
    }

    scci
  end # end calcCountsGSCCI

  def calcCountsSCCI(cwv, counts)
    if cwv == nil || counts == nil
      return nil
    end

    numGCNCodons = 0
    scci = 1.0
    numGCNCodons = counts.sum

    counts.each_index{
      |cIndex|
      if codonIndex(cIndex) != nil
        for i in 0 ... counts[cIndex]
          # scci = scci * ( cwv[cIndex] ** (1.0/numGCNCodons) )
          scci = scci * (cwv[cIndex])
        end
      end
    }

    #scci = scci/(100 * numGCNCodons)
    scci = scci ** (1.0/numGCNCodons)

    scci
  end


  # over the genes in the reference set, each codon weight is its
count/the count of the preferred
  #  codon in that family
  def calcWeights(refset)
    if refset == nil
      return nil
    end

    cc = Array.new(64, 0)
    cw = Array.new(64, 0)

    # Count codons over refset
    @gn.glist.each_index{
      |geneIndex|
      if refset[geneIndex] == 0
```

```
        next
      end

      for codon in 0 ... 64
        cc[codon] += @gn.glist[geneIndex].codonCounts[codon]
      end # end codon for
    } # end glist each

    # Calculate weight as count/maximal sibling
    for fam in 0 ... getFamCount
      famCodons = groupToCodons(fam)

      if famCodons == nil
        next
      end

      max = 0

      # Find maximum in family
      famCodons.each{
        |codonIndex|
        if cc[codonIndex] > max
          max = cc[codonIndex]
        end
      }

      max = max.to_f

      # Codon weight = count/maximal sibling
      famCodons.each{
        |codonIndex|
        if cc[codonIndex] > 0
          cw[codonIndex] = (cc[codonIndex].to_f / max)
        else
          cw[codonIndex] = 0.01
        end
      }
    end # end fam for

    # Each codon not in a family has its weight set to 1
    for codon in 0 ... 64
      if codonToGroup(codon) == nil
        cw[codon] = 1.0
      end
    end

    cw
  end # end calcWeights


  # Return a binary vector indicating the newSize genes with the
highest gSCCI value
  def findRefSet(size)
    geneIndex = Array.new(@gn.glist.size, 0)

    for i in 0 ... geneIndex.size
      geneIndex[i] = i
    end

    # sort genes (array of gene indices) by cai value
    for pos in 0 ... geneIndex.size
      for i in pos+1 ... geneIndex.size
        if @gn.glist[geneIndex[i]].SCCI >
@gn.glist[geneIndex[pos]].SCCI
          temp = geneIndex[pos]
          geneIndex[pos] = geneIndex[i]
          geneIndex[i] = temp
        #~ elsif @gn.glist[geneIndex[i]].SCCI ==
@gn.glist[geneIndex[pos]].SCCI
          #~ puts "Equal SCCI's: #{@gn.glist[geneIndex[i]].name} &
#{@gn.glist[geneIndex[pos]].name}"
          #~ puts "#{@gn.glist[geneIndex[i]].SCCI} =
#{@gn.glist[geneIndex[pos]].SCCI}"
        end
      end
    end

    #test
    #~ count = 0
    #~ geneIndex.each{
      #~ |index|
      #~ puts gn.glist[index].SCCI
      #~ count += 1
      #~ if count > size
        #~ puts "refset stop here"
      #~ end
    #~ }

    # refset vector indicates the top newSize genes
    refset = Array.new(@gn.glist.size, 0)

    for i in 0 ... size
      refset[geneIndex[i]] = 1
    end

    refset
  end # end findRefSet
end # end Bias module
```

Appendix B.   Perl scripts

B.1.   getGenes.pl

Developed by Raiford (Raiford 2005), this script extracts gene sequences from a Genbank annotated file.

```
$|=1;
#-------------------------------------------------------------------
#     The purpose of this program is to extract gene information from
#     an annotated complete genome file downloaded from the gene
#     bank.
#
#     We are only interested in protein genes because we will be
#     working with codons. For this reason the program scans the
#     annotated portion of the file looking for the CDS keyword. From
#     the CDS line we collect the start and stop location of the gene
and
#     extract the string of nucleotides from the sequence portion of
the
#     file.
#
#     Occasionally there is a frame shift indicated by a "join"
statement
#     on the CDS line. The program collects the various strings
#     indicated in the join statement and concatenates them together.
#     Also, occasionally, the CDS line indicates that the gene is on
the
#     other strand with the key word "complement" in which case we
#     take the reverse complement of the gene sequence.
#
#     Every gene is tested against the amino acid protein sequence
#     embedded within the annotation. Additionally,
#     checked that they are evenly divisible by three.
#-------------------------------------------------------------------

use strict;
use warnings;
#use RefineData;

#~ new method uses library. All adjustments have been made to that code
#~ will keep this code for a while to ensure that nothing is lost
my @argList = @ARGV;

#RefineData::getGenes(@argList);
#exit;

# declare and initialize variables
my @annotation = (  );  #storage for first half of genbank file
my $annotation = '';    #same but in one big string format
my $sequence = '';      #storage for second half (sequence data)
my $fileroot = "";                  #root of in and out file passed in
as arg
                        #store in and out filenames

my $unculledFlag=0;
my $filename = "";
my $outFileSuffix = "";
my $outputfile = "";
my $errorFile = "";
my $MINLEN = 306;
my $shortGene = 0;                  #counter to track number of short
genes
my $noteqGene = 0;
my $firstNum;           #each gene has a starting and ending loc
my $secNum;
my $numPhages = 0;
my %genetic_code;                   #hash to lu aa
my $numArgs = 0;                    #
my $NOTHREE = 0;                    #used as a flag to cull non
divisible by 3s (cull if 1)
my $NOPHAGE = 0;                    #used as a flag to cull phages
(don't cull unless tell)
my $NOEQ    = 0;                    #used as a flag to cull non equals
my $errorMessage = "";
my $tot=0;                                      #counter used to track
total number of CDS's

#can have nothree, nophage, noeq, len num, followed by file name
#loop through all args finding matches and setting Gs
#if ever find an odd ball then exit with a help screen
#get length of argv, last one should be file name
$numArgs = @ARGV;

$errorMessage=
"syntax should be
      perl getGenes.pl -nophage -nothree -len 100 -noeq -outfn suffix
fileroot
   or
      perl getGenes.pl -nocull fileroot
\n
nophage culls phage relateds
nothree will cull genes that are not divisible by three
len NUM indicates minimum number of codons (throw out any with smaller
number,
                       start and stop codons are automatically
accounted for)
noeq culls genes that do not equal the supplied protein seq. Usually
this
                       is because a stop codon shows up in the middle,
                                                                     or
some other unusual codon occurs.\n\n";

if($numArgs == 0){
      print $errorMessage;
      exit;
}

#last arg is filename so decrement numargs and it will index fileroot
and work
#for < nomenclature in for loop
$numArgs--;
$fileroot = $ARGV[$numArgs]."\n";
chomp($fileroot);
$filename = $fileroot.".in";
for(my $i=0; $i<$numArgs; $i++){
      print "argument $i is $ARGV[$i] \n";
      if($ARGV[$i] eq "-nothree"){
            $NOTHREE = 1;
      }
      elsif($ARGV[$i] eq "-nophage"){
            $NOPHAGE = 1;
      }
      elsif($ARGV[$i] eq "-nocull"){
            $unculledFlag = 1;
      }
      elsif($ARGV[$i] eq "-noeq"){
            $NOEQ = 1;
      }
      elsif($ARGV[$i] eq "-len"){
            $MINLEN = $ARGV[$i+1]*3+6;#times 3 for codons, the six
takes care of stop

                  #codons
            $i++; #this accounts for next argument which is the length
      }
      elsif($ARGV[$i] eq "-outfn"){
            $outFileSuffix=$ARGV[$i+1];
            $i++; #this accounts for the next argument which is the
file suffix
      }
      else{
            print $errorMessage;
            exit;
      }
}

$outputfile = $fileroot.$outFileSuffix.".out";
$errorFile = $fileroot.$outFileSuffix.".err";

unless( open(GET_FILE_DATA, $filename) ) {
    print STDERR "Cannot open file \"$filename\"\n\n";
    exit;
    }
open(OUTPUTFILE, ">$errorFile");#open it once to ensure empty
open(OUTPUTFILE, ">$outputfile");
                        #open input file
print "files open: ".$outputfile." ".$filename."\n";
my @filedata = <GET_FILE_DATA>;#get all data from input file
close GET_FILE_DATA;

my $in_sequence = 0;    #used as flag to determine when have entered
                        #sequence data portion of file
foreach my $line (@filedata) {
    if( $line =~ /^\/\/\n/ ) { # If $line is end-of-record line //\n,
        last; #break out of the foreach loop.
    } elsif( $in_sequence) { # If we know we're in a sequence,
            $sequence .= $line; # add the current line to $$dna.
    } elsif ( $line =~ /^ORIGIN/ ) { # If $line begins a sequence,
            $in_sequence = 1; # set the $in_sequence flag.
    } else{ # Otherwise
            push( @annotation, $line);  #add the current line to
@annotation.
    }                                                  #and seq, its in
BeginPerl use
}

$sequence =~ s/[\s0-9]//g;#extract white space out of sequence
```

```
my $seqFile = $fileroot."pureSeq.txt";
open(SEQFILE, ">$seqFile");#open it once to ensure empty
print SEQFILE $sequence."\n";

close(SEQFILE);

my $i=0;                #count of genes found
my $numNot3=0;          #count of genes that are not divisible by three
$annotation = join("",@annotation);#turn array into one big string
buildResList();         #needed when looking up codons, populates
                                                    #global var
genetic_code


#while( $annotation =~ /     CDS.*?gene="(.*?)"/sgm ) {#look for genes
#the above broke down in thermo cause it sometimes used /gene and
#sometimes used /locus_tag

while( $annotation =~ /     CDS.*?\/translation="/sgm ) {#look for
genes
      $tot++;
      my $complement = 0; #flag used to tell if the gene is comp or
not.
      my $join = 0;                 #flag used to tell if encountered a
join
      my $value = $&;
      $value =~ s/\%/perc_sign/;
      #now value has everything from CDS to beginning of sequence
      #print $value."\n--------\n";
      #print $geneName."\n---------------------------------\n\n";

      my $geneName="";
      if($value =~ /locus_tag="(.*?)"/){#do this for locustag
      #if($value =~ /gene="(.*?)"/){#do this for regular gene name
            $geneName=$1;
      }
      elsif($value =~ /gene="(.*?)"/){#do this second if look for locus
first
      #elsif($value =~ /locus_tag="(.*?)"/){
            $geneName=$1;
      }
      elsif($value =~ /standard_name="(.*?)"/){#do this second if look
for locus first
            $geneName=$1;
      }
      elsif($value =~ /note="(.*?)"/){#do this second if look for locus
first
            $geneName=$1;
      }
      elsif($value =~ /protein_id="(.*?)"/){#do this second if look for
locus first
            $geneName=$1;
      }
      else{
            print "ah oh, no gene name\n";
            #print $value."\n";
            #exit;
      }
      #print $geneName."\n";


  if($value =~ /complement/){
        $complement = 1;#it is the complement
  }

      if($value =~ /join\(/){
            $join = 1;
      }

  my $gene="";
      if(!$join){
    $value =~ /[0-9]+/gm;#get start and end locations for gene
    $firstNum = $&;
    $value =~ /[0-9]+/gm;
    $secNum = $&;
    #go get the sequence beginning at the first num and ending at sec
        $gene = substr($sequence, $firstNum-1, $secNum-$firstNum+1);
        #print $firstNum."  ".$secNum."\n";
  }

      else{
            print "there was a join\n";
    $value =~ /join\(.*?\)/s;
    my $joinLine = $&;
    print $joinLine."\n";
    $joinLine =~ /[0-9]+/gm;#get start and end locations for gene
    $firstNum = $&;
    $joinLine =~ /[0-9]+/gm;
    $secNum = $&;
    #go get the sequence beginning at the first num and ending at sec
        $gene = substr($sequence, $firstNum-1, $secNum-$firstNum+1);
        #print $firstNum."  ".$secNum."\n";
    while($joinLine =~ /,/gm){
                  $joinLine =~ /[0-9]+/gm;#get start and end locations
for gene
                  $firstNum = $&;
                  $joinLine =~ /[0-9]+/gm;
                  $secNum = $&;
      #print $firstNum."  ".$secNum."\n";

                  if( !($firstNum =~ /[0-9]+/) || !($secNum =~ /[0-
9]+/)) {
                        print " ah oh \n"."first num ".$firstNum."\n";
                        print " ah oh \n"."sedonc num ".$secNum."\n";
                        print "gene is $gene \n";

                        exit;
                        }
      #go get the sequence beginning at the first num and ending at sec
      $gene = $gene.substr($sequence, $firstNum-1, $secNum-
$firstNum+1);
            }
      }
      if ($complement){
            #print "there was a complement\n";
            $gene = reverse $gene;
    $gene =~ tr/ACGTacgt/TGCAtgca/;#complement
    }


      #print $gene."\n";
      my $phageCheckStr=$value;
      my $phageFlag = 0;

      if (  $phageCheckStr =~ /phage/i ||
                        $phageCheckStr =~ /virus/i ||
                        $phageCheckStr =~ /viral/i ||
                        $phageCheckStr =~ /transpos/i ){
                  #print "found a phage\n";
                  open(OUTPUTFILE, ">>$errorFile");
                  print OUTPUTFILE $phageCheckStr."\n";
                  print OUTPUTFILE "---------------------\n";
                  close(OUTPUTFILE);
                  open(OUTPUTFILE, ">>$outputfile");
                  if($NOPHAGE){
                        $numPhages=$numPhages+1;
                        $phageFlag = 1;
                  }
      }

      my $shortFlag = 0;
      if(length($gene)<$MINLEN){
            if(!$phageFlag) {$shortGene = $shortGene+1;}
            $shortFlag = 1;
      }

      my $noteqFlag=0;
      my $not3Flag=0;
      if(!$shortFlag    && !$phageFlag){
            #should be a good gene so go ahead and check for even
division
            if ( (length($gene)%3!=0) ){
                  open(OUTPUTFILE, ">>$errorFile");
                  printf OUTPUTFILE "modulus of gene ".$i."
".(length($gene)%3)."\n";
                  printf OUTPUTFILE "gene name is ".$geneName."\n"."---
------"."\n";
                  close(OUTPUTFILE);
                  open(OUTPUTFILE, ">>$outputfile");
                  if($NOTHREE){
                        $numNot3=$numNot3+1;
                    $not3Flag=1;
                  }
            }           #ok, got gene, now check against protein
                         #convert nucs to residues

            #now it is time to throw away the start codon
            #~ $gene = substr($gene,3,length($gene)-3);
            my $protein = getSeq($gene);        #get real seq from
annotation
    #~ print $protein."\n";

            $annotation =~ /[A-Z\s]*/sg;
            my $realProt = $&;
    #~ print $realProt."\n"; exit;
            $realProt =~ s/\s//g;
            #~ $realProt = substr($realProt,1,length($realProt)-
1);#start at 1 and go to


                        #one less than len to


                        #get rid of start codon


            #compare to my version
            if($protein ne $realProt){
                  if($NOEQ){
                        $noteqFlag=1;#set not equal flag to true
                        if(!$shortFlag && !$phageFlag &&
!$not3Flag){$noteqGene=$noteqGene+1;}
                  }

                  #print $protein."\n\n\n".$realProt; exit;
                  open(OUTPUTFILE, ">>$errorFile");
 printf OUTPUTFILE "Ah oh, my seq did not match the REAL protein:
".$geneName."\n";
                  #print $gene."\n\n\n";
                  #print $protein."\n\n\n";
                  #print $realProt."\n";
                  my $compIdx=0;
                  while(      substr($realProt,$compIdx,1)
                                          eq
                              substr($protein,$compIdx,1)){
                        $compIdx=$compIdx+1;
                  }

                  printf OUTPUTFILE "strings are different at loc
".$compIdx."\n";
                  printf OUTPUTFILE "the nuc seq was
".substr($gene,$compIdx*3,3)."\n";
                  printf OUTPUTFILE "the aa was
".substr($realProt,$compIdx,1)."\n";
      printf OUTPUTFILE "---------------\n";
                  close(OUTPUTFILE);
                  open(OUTPUTFILE, ">>$outputfile");
            }
      }

  if($unculledFlag){#set these flags to false if do not wish to cull
            $noteqFlag = 0;
            $shortFlag = 0;
            $phageFlag = 0;
      }
      if( $geneName =~ /operon/ ){
            print "got an operon $geneName \n";
            exit;
      }
      if( $geneName =~ /operon/ ||  $noteqFlag || $shortFlag ||

                        $phageFlag || $not3Flag)   {
            #do not output or increment
      }

      else{
            $i=$i+1;           #increment gene count but only if it is a
gene
            #printf  "<%s>\n%s\n",$geneName,$gene;
            printf OUTPUTFILE "<%s>\n%s\n",$geneName,$gene;
      }

      if($i%1000==0 && $i!=0){
      print $i."\n";  #every thousand genes send output to screen so we
                              #know the prog is alive
      }

}
#output to screen total num of genes
print "Total number of genes (after culling)               ".$i."\n";
if($NOTHREE){
      print "Total number of genes that are not divisible by 3
".$numNot3."\n";
}
print "Total number of short genes
".$shortGene."\n";
if($NOPHAGE){
      print "Total number of phage relateds
".$numPhages."\n";
}
if($NOEQ){
      print "Total number of not equals
".$noteqGene."\n";
}
print   "Total number before culling
".$tot."\n";
close(OUTPUTFILE);

open(OUTPUTFILE, ">>$errorFile");
print OUTPUTFILE "Total number of genes
".$i."\n";
if($NOTHREE){
      print OUTPUTFILE "Total number of genes that are not divisible by
3   ".$numNot3."\n";
}
print OUTPUTFILE "Total number of short genes (not included)
".$shortGene."\n";
if($NOPHAGE){
      print OUTPUTFILE "Total number of phage relateds
".$numPhages."\n";
}
```

```
if($NOEQ){
      print OUTPUTFILE "Total number of not equals
".$noteqGene."\n";
}
print   OUTPUTFILE "Total number before culling
".$tot."\n";

close(OUTPUTFILE);
exit;

sub getSeq
{
      my ($passedSeq) = @_;
      my $codon = '';
      my $residue = '';
      my $len = length($passedSeq);
      my $num = 0;
      my $protein = '';
      $passedSeq =~ s/s/g/; #get rid of quotes
      $passedSeq =~ s/r/g/; #get rid of quotes
      $passedSeq =~ s/y/t/; #get rid of quotes
      $passedSeq =~ s/n/t/; #get rid of quotes
      $passedSeq =~ s/m/a/; #get rid of quotes
      #for each codon
      for ($num = 0; $num<$len-2; $num = $num+3)
      {
            #convert to residue
            $codon = substr($passedSeq, $num, 3);

            $residue = getRes($codon);
            #concat with growing protein
            if($residue ne "_" && $num != 0){
                  $protein = $protein.$residue;
    }elsif($num == 0){
      $protein = $protein."M";
    }
  }
            return $protein;

}

sub getRes
{
      my($codon) = @_;  $codon = uc $codon;#converts to uppercase

    if(exists $main::genetic_code{$codon}) {
        return $main::genetic_code{$codon};
    }    else    {
                  print STDERR "Bad codon \"$codon\"!!\n";
                  return '-';
    }
}

sub buildResList{
    %main::genetic_code = (

    'TCA' => 'S',    # Serine
```

```
'TCC' => 'S',     # Serine
'TCG' => 'S',     # Serine
'TCT' => 'S',     # Serine
'TTC' => 'F',     # Phenylalanine
'TTT' => 'F',     # Phenylalanine
'TTA' => 'L',     # Leucine
'TTG' => 'L',     # Leucine
'TAC' => 'Y',     # Tyrosine
'TAT' => 'Y',     # Tyrosine
'TAA' => '_',     # Stop
'TAG' => '_',     # Stop
'TGC' => 'C',     # Cysteine
'TGT' => 'C',     # Cysteine
'TGA' => '_',     # Stop
'TGG' => 'W',     # Tryptophan
'CTA' => 'L',     # Leucine
'CTC' => 'L',     # Leucine
'CTG' => 'L',     # Leucine
'CTT' => 'L',     # Leucine
'CCA' => 'P',     # Proline
'CCC' => 'P',     # Proline
'CCG' => 'P',     # Proline
'CCT' => 'P',     # Proline
'CAC' => 'H',     # Histidine
'CAT' => 'H',     # Histidine
'CAA' => 'Q',     # Glutamine
'CAG' => 'Q',     # Glutamine
'CGA' => 'R',     # Arginine
'CGC' => 'R',     # Arginine
'CGG' => 'R',     # Arginine
'CGT' => 'R',     # Arginine
'ATA' => 'I',     # Isoleucine
'ATC' => 'I',     # Isoleucine
'ATT' => 'I',     # Isoleucine
'ATG' => 'M',     # Methionine
'ACA' => 'T',     # Threonine
'ACC' => 'T',     # Threonine
'ACG' => 'T',     # Threonine
'ACT' => 'T',     # Threonine
'AAC' => 'N',     # Asparagine
'AAT' => 'N',     # Asparagine
'AAA' => 'K',     # Lysine
'AAG' => 'K',     # Lysine
'AGC' => 'S',     # Serine
'AGT' => 'S',     # Serine
'AGA' => 'R',     # Arginine
'AGG' => 'R',     # Arginine
'GTA' => 'V',     # Valine
'GTC' => 'V',     # Valine
'GTG' => 'V',     # Valine
'GTT' => 'V',     # Valine
'GCA' => 'A',     # Alanine
'GCC' => 'A',     # Alanine
'GCG' => 'A',     # Alanine
'GCT' => 'A',     # Alanine
'GAC' => 'D',     # Aspartic Acid
'GAT' => 'D',     # Aspartic Acid
```

```
    'GAA' => 'E',     # Glutamic Acid
    'GAG' => 'E',     # Glutamic Acid
    'GGA' => 'G',     # Glycine
    'GGC' => 'G',     # Glycine
    'GGG' => 'G',     # Glycine
    'GGT' => 'G',     # Glycine
    );

}
```

Appendix C. MATLAB toolboxes and commands

The MATLAB Statistical Pattern Recognition Toolbox provided the implementation of PCA used for the research described in this thesis. With this toolbox, PCA and data projection are accomplished using the following sequence of commands.

```
model = pca(X,2);
projected_data = linproj(X,model);
```

Bibliography

Bennetzen, J. and B. Hall (1982). "Codon selection in yeast." Journal of Biological Chemistry **257**(6): 3026-3031.

Bulmer, M. (1991). "The selection-mutation-drift theory of synonymous codon usage." Genetics **129**(3): 897-907.

Carbone, A., F. Kepes, et al. (2005). "Codon bias signatures, organization of microorganisms in codon space, and lifestyle." Mol Biol Evol **22**(3): 547-61.

Carbone, A., A. Zinovyev, et al. (2003). "Codon adaptation index as a measure of dominating codon bias." Bioinformatics **19**(16): 2005-15.

Chargaff, E. (1950). "Chemical specificity of nucleic acids and mechanism of their enzymatic degradation." Experientia **6**(6): 201-9.

Foerstner, K. U., C. von Mering, et al. (2005). "Environments shape the nucleotide composition of genomes." EMBO Rep **6**(12): 1208-13.

Freire-Picos, M. A., M. I. Gonzalez-Siso, et al. (1994). "Codon usage in Kluyveromyces lactis and in yeast cytochrome c-encoding genes." Gene **139**(1): 43-9.

Gouy, M. and C. Gautier (1982). "Codon usage in bacteria: correlation with gene expressivity." Nucleic Acids Research **10**(22): 7055.

Grantham, R. (1980). "Workings of the genetic code." Trends Biochem. Sci **5**: 327-331.

Grantham, R., C. Gautier, et al. (1981). "Codon catalog usage is a genome strategy modulated for gene expressivity." Nucleic Acids Res **9**(1): r43-74.

Grantham, R., C. Gautier, et al. (1980). "Codon catalog usage and the genome hypothesis." Nucleic Acids Research **8**(1): 197.

Heizer, E. M., Jr., D. W. Raiford, et al. (2006). "Amino acid cost and codon-usage biases in 6 prokaryotic genomes: a whole-genome analysis." Mol Biol Evol **23**(9): 1670-80.

Hinds, P. and R. Blake (1985). "Delineation of coding areas in DNA sequences through assignment of codon probabilities." Journal of Biomolecular Structure and Dynamics **3**(3): 543-550.

Hotelling, H. (1933). "Analysis of a complex of statistical variables into principal components." Journal of Educational Psychology: 498-520.

Ikemura, T. (1981). "Correlation between the abundance of Escherichia coli transfer RNAs and the occurrence of the respective codons in its protein genes." J Mol Biol **146**(1): 1-21.

Ikemura, T. (1981). "Correlation between the abundance of Escherichia coli transfer RNAs and the occurrence of the respective codons in its protein genes: a proposal for a synonymous codon choice that is optimal for the E. coli translational system." J Mol Biol **151**(3): 389-409.

Ikemura, T. (1985). "Codon usage and tRNA content in unicellular and multicellular organisms." Mol Biol Evol **2**(1): 13-34.

Kanaya, S., Y. Yamada, et al. (1999). "Studies of codon usage and tRNA genes of 18 unicellular organisms and quantification of Bacillus subtilis tRNAs: gene expression level and species-specific diversity of codon usage based on multivariate analysis." Gene **238**(1): 143-55.

Konigsberg, W. and G. Godson (1983). "Evidence for use of rare codons in the dnaG gene and other regulatory genes of Escherichia coli." Proceedings of the National Academy of Sciences **80**(3): 687-691.

Krane, D. E. and M. L. Raymer (2003). Fundamental concepts of bioinformatics. San Francisco, Benjamin Cummings.

Lafay, B., A. T. Lloyd, et al. (1999). "Proteome composition and codon usage in spirochaetes: species-specific and DNA strand-specific mutational biases." Nucleic Acids Res **27**(7): 1642-9.

McLachlan, A. D., R. Staden, et al. (1984). "A method for measuring the non-random bias of a codon usage table." Nucleic Acids Res **12**(24): 9567-75.

Moran, N. A. (2002). "Microbial minimalism: genome reduction in bacterial pathogens." Cell **108**(5): 583-6.

NHGRI. (2009). "genome.gov @ National Human Genome Research Institute." Retrieved May 12, 2009, from http://www.genome.gov.

Raiford, D. W. (2005). Multivariate analysis of prokaryotic amino acid usage bias: a computational method for understanding protein building block selection in primitive organisms, Wright State University. **Master of Science**.

Raiford, D. W., D. E. Krane, et al. (2008). "Automated isolation of translational efficiency bias that resists the confounding effect of GC(AT)-content." IEEE/ACM Transactions on Computational Biology and Bioinformatics.

Rocha, E. P. and A. Danchin (2002). "Base composition bias might result from competition for metabolic resources." Trends Genet **18**(6): 291-4.

Sharp, P., E. Bailes, et al. (2005). "Variation in the strength of selected codon usage bias among bacteria." Nucleic Acids Research **33**(4): 1141.

Sharp, P. M. and W. H. Li (1987). "The codon Adaptation Index--a measure of directional synonymous codon usage bias, and its potential applications." Nucleic Acids Res **15**(3): 1281-95.

Sharp, P. M., T. M. Tuohy, et al. (1986). "Codon usage in yeast: cluster analysis clearly differentiates highly and lowly expressed genes." Nucleic Acids Res **14**(13): 5125-43.

Shields, D. C., P. M. Sharp, et al. (1988). ""Silent" sites in Drosophila genes are not neutral: evidence of selection among synonymous codons." Mol Biol Evol **5**(6): 704-16.

Smith, L. I. (2002). A tutorial on Principal Components Analysis.

Sueoka, N. (1962). "On the genetic basis of variation and heterogeneity of DNA base composition." Proc Natl Acad Sci U S A **48**: 582-92.

Varenne, S., J. Buc, et al. (1984). "Translation is a non-uniform process. Effect of tRNA availability on the rate of elongation of nascent polypeptide chains." J Mol Biol **180**(3): 549-76.

Vetsigian, K. and N. Goldenfeld (2009). "Genome rhetoric and the emergence of compositional bias." Proc Natl Acad Sci U S A **106**(1): 215-20.

Wright, F. (1990). "The 'effective number of codons' used in a gene." Gene **87**(1): 23-9.