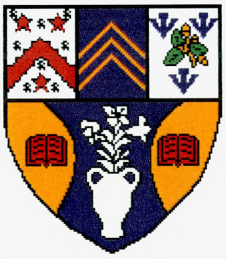


UNIVERSITY of ABERTAY DUNDEE

Genes and Protein Structural Elements are Delineated by DNA Mirror Repeats and Characterized by Net Dinucleotide Composition

Ph.D. Thesis
University of Abertay Dundee
Dundee, Scotland

May, 2009

Dorothy MacLean Lang
B.S., M.S. State University of New York at Albany, USA
M.S. Bioinformatics, University of Abertay Dundee
[redacted]

This report, together with four related published papers,
fulfill the publication requirements of a Thesis-By-Paper for the
Ph.D. Research Degrees Program at the University of Abertay-Dundee.

Genes and Protein Structural Elements are Delineated by DNA Mirror Repeats and Characterized by Net Dinucleotide composition

Dorothy MacLean Lang

A thesis submitted in partial fulfilment of the
requirements of the University of Abertay Dundee
for the degree of Doctor of Philosophy

May 2009

I certify that this thesis is the true and accurate version of the thesis approved by the examiners.

Signed ... [redacted]
(D

Date...14/10/09

Contents

Abbreviations

CA circuit assemblage

IMR imperfect mirror repeat

IR inverted repeat

MR mirror repeat

mIMR maximal imperfect mirror repeat

rdIMR reverse dinucleotide terminated imperfect mirror repeat

PSE protein structural elements - helices, β–sheets, turns, etc.

SNPS single nucleotide polymorphisms

TNR trinucleotide repeat

Abstract

Dinucleotide composition and sequence repeats are attributes of DNA that have been used for more than 40 years to evaluate sequence identity and function, phylogeny and evolution. This thesis provides new perspectives of dinucleotide composition and DNA organization by the development and application of several novel methods of analyzing sequence composition and repeats. The key aspects of the new methods described in this thesis are the identification of "net dinucleotide counts", i.e. rXY - sYX = (r-s) netXY, and the systematic identification of imperfect mirror repeats, i.e. sequence segments that have an axis of symmetry on a single strand and a position determined by sequential evaluation of a sequence from 5' to 3'. These measurements were found to be different perspectives of related components. For example, the segment ACGTGCA is a perfect mirror repeat with a net dinucleotide value of zero. Analysis of the distribution of imperfect mirror repeats (IMRs) found that IMRs coincide with protein structural elements. Therefore, the net dinucleotide value of a sequence is a measure of those dinucleotides that are not principally associated with the span of protein structural elements. It was found that many sequences consist of a limited number of net dinucleotides that could readily be combined - e.g., 2nAC + 2nCT + 2nTA = 2nACTA. Each of these combined values is called a circuit, and has both a qualitative and quantitative component. If a sequence consists of several circuits, they are collectively referred to as the circuit assemblage of the sequence. Circuits and circuit assemblages were found to differentiate phylogeny and function in a manner consistent with traditional phylogenetic and functional classification at all levels of genome organization. Analysis of mirror repeats and PSEs reveals that genes and genomes are hierarchically ordered, that the hierarchal order is based on the distribution of reverse dinucleotide pairs and sequence repeats, that the span of protein structural elements (helices, sheets and turns) coincides with DNA mirror repeats, and that many key aspects of protein function can be inferred from repeat motifs in the DNA from which it is translated.

Overview

The studies constituting this thesis are a spectrum of systematic computational analyses designed to investigate an observation by the author - that protein motifs appeared to be the translated product of DNA imperfect mirror repeats (IMRs) - and the serendipitous discovery that the *net* dinucleotide composition of *E. coli* tRNAs provides a unique succinct identity for each tRNA species. The uniqueness of tRNA net dinucleotides suggested net dinucleotides might also reveal functional relatedness in protein-coding DNA.

Simplistic attempts to describe the relationship of IMRs to protein structural elements (PSEs) - helices, β–sheets, β–strands and turns - were foiled by the multiple numbers of IMRs that occur in protein-coding DNA. Therefore, systematic methods were developed to map and characterize two types of imperfect mirror repeats (IMRs) - the longest or maximal within a sequence (mIMRs) and IMRs identified based on the nearest downstream reverse dinucleotide (rdIMRs). The coincidence of IMRs with PSEs was evaluated for various degrees of symmetry, and ≥ 50% symmetry was found to be an informative cut-off for all of the proteins evaluated for this property (N=17). Both mIMRs and rdIMRs were mapped to the proteins for which the tertiary structure and constituent secondary structural elements had been determined experimentally. Short IMRs were progressively removed from the IMR set of each protein until the segments remaining no longer coincided with PSEs with statistical significance. This procedure identified the essential minimum IMR value for each protein at which IMRs coincided with PSEs. The length of the essential minimum IMR value of each protein was most frequently in the range of 17-20 nt. Interestingly, the essential minimums are within the same range as the minimum sequence length required to identify unique segments of human sequences (Kerr and Workman, 1994) and the optimal length for creating primers for sequencing most organisms (http://www.eppendorf.de/int/index. php?l =131&action= products&contentid=109&sitemap=2.5.3.6). RdIMRs were found to map to PSEs and mIMRs to map to protein functional domains.

While evaluating IMRs in protein-coding DNA, it became apparent that there was also a high frequency of short mirror repeats (e.g. ACA, ACCA). This observation led to

the idea of combining dinucleotides and their reverse (e.g., CG and GC) to obtain a novel measurement of "net dinucleotides" (e.g., nCG). The first sequences analyzed had a limited number of net dinucleotides that were equal in number, which readily lent itself to be group into a single expression, which was named a "circuit."

e.g., 2nAC + 2nCT + 2nTA (net dinucleotides) => 2nACTA (circuit)

As more sequences were evaluated, many contained several combinations of circuits. All circuits derived from a sequence were identified as its circuit assemblage (CA) - a measurement of the number and composition of clusters of net dinucleotides of equal value for any sequence. Although net dinucleotides can be evaluated separately (rather than combined into circuits), it is as circuits that they exhibit functional discrimination at all levels of genomic organization.

IMRs and CAs are both extensions of traditional methods of DNA analysis. Mirror repeats have been described periodically in the literature, usually associated with a particular functional aspect of a protein, but no systematic and comprehensive evaluation of this feature in protein-coding DNA has been made previously. The dinucleotide composition of DNA has been used as a descriptor since 1934, and has long been established as being function- and species-specific. *Net* dinucleotide composition, however, has never previously been evaluated.

The methods described in these studies provide new insights into evolution, speciation and gene function, and several tools to improve the prediction of significant functional sites in proteins. They are based on intrinsic properties of the sequence - composition combined with order (dinucleotides) - and do not require the alignment of related sequences. The IMR computational methods described can be used to identify key functional sites in the translated protein and to unequivocally determine the span of PSEs. CAs provide a consise sequence signature related to the function of the segment - which varies based on the level of genomic organization (PSEs, gene, operon, organism). CAs are shown (in a paper published as part of this thesis) to be consistent with traditional phylogenetic methods (tree-building), despite the fact that they are not based on sequence alignment. This suggests they may also be useful for the classification of familial protein groups within a species, identification of the familial association of hypothetical and unknown proteins, and pathway analysis. IMRs and CAs are related by mutual constraint.

Most PSEs consist of IMRs - i.e. a span consisting of dinucleotides and their reverse dinucleotides. Dinucleotides that are not components of PSEs contribute to the net dinucleotide count - circuits. IMRs, then, represent the relatively stable PSEs, and CAs the dinucleotides associated with the configuration of the more variable spans that determine the tertiary structure of the protein.

1. Introduction

This thesis describes several new methods for the analysis of DNA sequences that provide new perspectives of the relationship between DNA, proteins, protein function and evolution. These methods are based on DNA features - dinucleotide composition and DNA repeats - that have long been recognized as significant to genome identity and protein function, but use these features in new and systematic contexts to derive novel measurements that lead to new insights. Mirror repeats have been described periodically in the literature, usually associated with a particular functional aspect of a protein, but no systematic and comprehensive evaluation of this feature in protein-coding DNA has been made previously. The dinucleotide composition of DNA has been described since 1934, and has long been established as being function- and species-specific. *Net* dinucleotide composition, however, has never previously been evaluated.

More than 50 years ago, Watson and Crick (1953) discovered that the genetic blueprint of all organisms consists of the linear arrangement of four types of deoxyribonucleic acids. The first complete genome was sequenced 13 years ago (Fleischmann, 1995). During the interim between these events, most DNA analysis consisted of the comparison of protein coding DNA from different organisms - comparisons of dinucleotide composition and sequence alignment. These strategies continue to be essential to comparative genomics, and are also a basis for the analyses described in this thesis.

Comparative analysis of proteins uses the same basic strategies as those used for DNA. The sequential order and pairwise comparison of amino acids are used to predict the secondary and tertiary structure, functional interactions and the evolutionary history of proteins. A large proportion of protein research is related to the identification of functional

motifs - which are also based on sequence order, though motifs are very short and often contain gaps. Identification of key functional sites within a protein, however, requires more than the solution of a linear problem because the function of a particular amino acid sequence or motif is impacted by its spatial position within the tertiary structure of the protein, which is often unknown.

To summarize, the basic strategies used for sequence analysis - for DNA and protein - have remained unchanged since its discovery. These strategies include: (a) analysis of nucleotide and dinucleotide composition, (b) analysis of pairwise alignment and its derivatives, (c) identification of functionally significant motifs. Multiple forms of each of these strategies continue to be developed and enhanced, to increase predictive competence or to analyze specific attributes of the genome. The methods described in this thesis are, similarly, based on these strategies, though novel extensions of them.

2. Background

Modern technology has simplified the determination of DNA and protein sequences and greatly increased the speed of processing them. Sequencing of DNA and proteins is taking place at academic, governmental and industrial laboratories, many of which contribute their results to databases that are accessible worldwide. One of these is the NCBI Genome Project, which currently contains the DNA and protein sequences for 1913 complete genomes (http://www.ncbi.nlm.nih.gov/ genomes/static/gpstat.html). These sequences and others within the other NCBI databases (http://www.ncbi.nlm.nih.gov/Database/index.html) provide abundant resources for sequence analysis. Information within the NCBI database is shared with the databases of the European Bioinformatics Institute EMBL-EBI (http://www.ebi.ac.uk/) and the Database of Japan (http://www.ddbj.nig.ac.jp/).

Understanding how sequences "generate" and "proliferate" life is now in most need of new and more informative methods. Current methods of sequence analysis are enhancements or modifications of methods that have been used since the original discovery of the role of DNA. Most of these methods are based on combined information about sequence composition and order. Large sequences such as genes and genomes are compared to each other and scored for similarity and differences. Sequence segments

generally less than twelve nucleotides or amino acids that have a characteristic composition and order and can be recognized in multiple organisms are referred to as sequence motifs. Many have a specific function. Short segments that are highly repetitive are another type of motif as their identity is based on a specific composition within a specific span. In the background information that follows, the main types of sequence analysis will be described briefly, and the relationship to the new methods developed during this thesis study will be described.

2a. Historical analysis of DNA composition

Nucleotide composition was one of the first attributes of DNA to be recognized as being highly specific to each species at all levels of genomic organization. Osawa (1960) identified the species-specificity of nucleotides in diverse organisms and Mukai (1965) subsequently demonstrated species-specificity in closely related organisms. Henney and Storck (1962) demonstrated that nucleotide composition characterized some cellular compartments within a single organism. Nucleotide composition has continued to be an essential reportable attribute of sequence analysis.

It took more than a decade for dinucleotide composition to also become recognized as highly specific to a species. One of the first studies was by Bernardi *et al* (1975), which compared the dinucleotide composition of deoxyribonuclease in 5 different organisms and found that both the dinucleotide composition of the protein and the 5'-terminal dinucleotides were distinctive for each organism. This study demonstrated that dinucleotide composition differentiated species at multiple levels of organization, as had previously been demonstrated for nucleotide composition only.

Ruth Nussinov was one of the first researchers to extensively study dinucleotide composition and DNA repeats, and published more than 100 papers on these topics. One of her first studies was nearest-neighbor (dinucleotide patterns) in both prokaryotic and eukaryotic cells (Nussinov, 1981). Other papers include algorithms for the discovery of DNA repeats (Nussinov, 1983), prediction of DNA secondary structure (Nussinov and Jacobson, 1980) and cross-species comparisons of the dinucleotide composition of selected vertebrates, non-vertebrates, DNA viruses, mitochondria, RNA viruses, bacteria and phage (Nussinov, 1984). Subsequently, her interest in patterns in DNA and amino acid sequences

turned to the identification of signal motifs (Nussinov, 1987). Nussinov did not pursue patterns in protein-coding DNA, which is one of the main objectives of the studies encompassed by this thesis. But her work established the relationship of dinucleotides to species identity - the point of embarkation for the studies of net dinucleotides contained in this thesis.

From 1983 through 1994, most studies of dinucleotide composition focused on investigation of GC content and its relationship to genomic islands, methylation and development. In 1994, Samuel Karlin introduced the concept of a genomic signature, which is a quantitative measure of the difference between the dinucleotide counts of two species. He applied the methodology to a study of the evolutionary relationships of the herpes virus (Karlin *et al*, 1994) and a broad spectrum of eukaryotes (Karlin and Ladunga, 1994). In subsequent studies he evaluated the genomic signature in diverse and comprehensive groups of organisms and organelles (Karlin, 1998; Karlin *et al*, 1998; Gentles and Karlin, 2001). Most of Karlin's studies evaluate the complete dinucleotide composition and do not differentiate between coding and non-coding DNA. The dinucleotide counts used for the genomic signature include the combined dinucleotide counts from each strand and its reverse complement. Karlin does not relate dinucleotide composition to protein functions except in selected instances such as genomic islands. The sequences analyzed in this thesis are based on specific functional entities that have an identified start and stop - tRNAs, ribosomal DNAs, genes and genomes. The new methods that are described are intrinsic quantitative measurements, rather than comparative. The derived values for each gene segment can be compared to others, but each is independent and therefore portable.

More recently, the relationship between sequence order and dinucleotide composition has been evaluated for both global and specific objectives. Forsdyke (1995) evaluated the affect of nucleotide composition and order on dinucleotide composition, finding that sequence order significantly impacted dinucleotide composition. Zhang and Zhao (2004) evaluated the affect of neighboring-nucleotide composition on single nucleotide polymorphisms (SNPs) in the mouse and human. Although these studies were directed towards a particular case of dinucleotide variation, this mechanism would undoubtedly be a factor in the development of dinucleotide profiles over evolutionary

time. Arndt and Hwa (2005) also investigated mechanistic processes that influence nearest-neighbor substitution, hence dinucleotide composition, and developed methods for making corrections for these factors in order to make more realistic models of evolution. Their hypothesis suggests that their perspective of dinucleotide neighbors is that mechanistic constraints related to observed dinucleotide changes detract from the "true nature" of evolution.

Nakashima *et al* (1998) demonstrated that all the protein coding genes within nine genomes - *Escherichia coli, Haemophilus influenzae, Helicobacter pylori, Mycoplasma genitalium, Mycoplasma pneumoniae, Synechocystis sp., Methanococcus jannaschii, Archaeoglobus fulgidus, and Saccharomyces cerevisiae* - could be distinguished by species on the basis of dinucleotide composition with 88% accuracy, thus demonstrating the inherent relationship between dinucleotide composition and species identity. This study also compared the dinucleotide composition of protein-coding DNA with whole genome DNA and found them to be highly similar. The analyses described in this thesis expand the discrimination identified by Nakashima *et al* (1998) to net dinucleotides, and clusters of net dinucleotides of equal value.

The fine-scale specificity of dinucleotide composition is also demonstrated by several recent studies. Schattner (2002) used dinucleotide composition to identify non-protein coding RNAs (ncRNAs) in three genomes - *Methanococcus jannaschii, Plasmodium falciparum and C. elegans* - with ≥ 97% accuracy. Andrieu *et al* (2004) used dinucleotide composition to detect transposable elements in three genomes - *Drosophila melanogaster, Caenorhabditis elegans and Arabidopsis thaliana.* Both of these studies demonstrate that fine-scale discrimination by dinucleotides is a global feature - common to diverse organisms. The analyses described in this thesis demonstrate that net dinucleotides also discriminate fine-scale functional variations in gene segments, by evaluating the CAs for each of the functional domains of the HIV-1 *env* protein (Lang, 2007b).

2b. Sequence alignment methods

Historically and currently, a segment of DNA or protein is identified by comparing it to known sequences. Based on this comparison, the identity of the species and the sequence position relative to the known sequence can be determined, and often some aspect of the

function can be inferred. The most frequently used methods of identification are the algorithms BLAST (Altchul *et al*, 1990) and WU-BLAST (http://blast.wustl.edu/). They both include blastp (protein-to-protein blast), tblastn (protein to 6-frame translated DNA blast), and blastn (DNA-to-DNA blast). Genomes or segments can also be blasted against themselves in order to identify repeated gene segments. Blast comparison services are available online at the National Center for Biotechnology Information (NCBI) in the United States (http://www.ncbi.nlm.nih.gov/) and the European Molecular Biology Laboratory (EMBL) in Europe (http://www.embl.org/), and other sites, or as downloadable files and applications.

The identification of similar genes or proteins facilitates the identification of the function of an unknown segment. The underlying assumption is that sequence similarity likely results in functional similarity. This provides a theoretical basis for inferring the function of genes within newly sequenced genomes. BLAST identifies a query sequence by making a pairwise comparison of it to other sequences contained within its database, which may also consist of a blast against itself. BLAST output consists of multiple pairwise alignments of two sequences or tabulated query-sequence pairs identified by "gi numbers" (sequence identifier numbers generated by the National Center for Biotechnology Information (NCBI)), that have sequence similarity greater than a cut-off set by the user. The BLAST measurement of sequence similarity is analogous to one of the features analyzed in this thesis - "imperfect mirror repeats." Matches between sequence segments are often imperfect, and the percent identity traditionally has been used to evaluate both functional and phylogenetic similarity.

In order to compare genes and genomes, more than 2 sequences are frequently aligned to each other. There are many multiple alignment programs available on the internet that were developed by different researchers at universities throughout the world. Two of the most frequently used and oldest are CLUSTAL W (Thompson *et al*, 1994) and MEGA (Kumar *et al*, 1994). MAVID (Bray and Pachter, 2003) is an on-line web server that accepts large alignments. The process of sequence alignment impacts the constituents of the set that is being aligned. The objective of an alignment is to place identical nucleotides of each sequence in the same vertical column. Gaps are introduced in order to "optimize" an alignment, i.e. make the most number of close matches in each vertical

column. DNA sequences may also be codon-aligned, i.e. gaps are added to an alignment of nucleotides to ensure that most of the sequence will translate the amino acids that are known to compose most of the set.

A phylogenetic tree is a quantitative measurement of the differences between sequences within a multiple alignment. There are at least 355 computer programs that can be used to build phylogenetic trees. These are listed and categorized on the website maintained by Joe Felsenstein at the University of Washington, Seattle, USA (http://evolution.genetics. washington.edu/phylip/software.html). Some of the common uses of multiple alignments and phylogenetic trees are to infer common ancestors, characterize subtypes of viruses, and evaluate transmission cases in both population studies and criminal investigations. Most tree-building programs require aligned sequences. Some tree-building methods strip columns containing gaps before computing the distance between the sequences. Others use sequences that have been codon aligned, a process that introduces additional gaps, which may then be stripped. To summarize, most phylogenetic relationships are based on modified sequences because the requisite sequence alignment introduces of gaps.

The methods that are developed in this thesis have a particular advantage in that they do not require sequence alignment, yet provide similar information to that derived from aligned sequences. Additionally, once the net dinucleotides of a sequence are determined, the sequence can be evaluated within any set of sequences.

It is demonstrated in two of the published papers (Lang 2000, 2007a) that are components of this thesis, that in ribosomal DNA and subgroups of viruses, net dinucleotides and CAs, identify the same phylogenetic groups as those identified by traditional phylogenetic analysis. This suggests that net dinucleotides and CAs capture intrinsic properties of the sequence including both its species identity and function.

2c. DNA composition and secondary and tertiary structure

DNA forms secondary structures (subject to highly specific environmental conditions) that are based on the linear arrangement of nucleotides of different composition. This is because the same molecular forces that hold the double helix together cause single strands of DNA or RNA to fold into various structures, including hairpins and cruciforms. DNA

secondary and tertiary structures are directly related to both the composition and internal symmetry of DNA repeats, because these features of a repeat either promote or disallow DNA and RNA folding. The DNA sequence (i.e. the linear arrangement of its components) determines which parts of a strand are available to interact. The cellular environment and mechanical aspects such as whether or not the strand may be tethered to a cell membrane or in the process of transport additionally affects DNA and RNA secondary and tertiary structure. Several websites are available that can be used to predict the theoretical secondary structure of a sequence - including "Vienna RNA Secondary Structure Prediction" at the University of Vienna, Austria (http://rna.tbi.univie.ac.at/cgi-bin/RNAfold.cgi) (Hofacker, 2003) and "Mfold" at Renssalaer Polytechnical Institute, NY, USA (http://frontend.bioinfo.rpi.edu/applications/ mfold/cgi-bin/rna-form1.cgi) (Zucker, 2003). A significant advantage of the methods described in this thesis is that it identifies protein segments that are significant to structure and function without knowledge of protein tertiary structure.

2d. Sequence repeats and motifs in DNA and proteins

The main types of sequence motifs are repeats and functional motifs, and both occur in DNA and protein. DNA repeats vary in length substantially, depending on the type of repeat and the criteria for identity. The types of repeats are illustrated below in **Figure 1.**

Figure 1 illustrates that palindromes (1c.) are self-complementary on a single strand, and mirror repeats (1d.) are self-complementary on the reverse strand. Therefore, RNA segments that are inverted repeats can fold back on themselves to form hairpins. Under certain conditions, a DNA segment can form a triplex by folding back to make a hairpin conformation with the complementary strand of the primary repeat unit (Sinden *et al*, 1994). The likelihood of folding is based on the percent identity of the strands that have the potential to fold, which theoretically decreases as the repeats become more imperfect.

a. forward repeat or direct repeat or tandem repeat

```
  ------> ------>
5-GACTAAG GACTAAG-3
3-ctgattc ctgattc-5
```

b. complement match

```
  ------>
5-GACTAAG ctgattc-3
3-ctgattc GACTAAG-5
          ------>
```

c. palindrome or inverted repeat

```
  ------>
5-GACTAAG cttagtc-3
3-ctgattc GAATCAG-5
          <------
```

d. mirror repeat or reverse match

```
  ------> <------
5-GACTAAG GAATCAG-3
3-ctgattc cttagtc-5
```

Figure 1. The four possible types of repeats are illustrated here for DNA. The same types potentially exist in proteins, although the length of each and the number of them within a sequence would be much smaller, and less perfect, than for DNA.

As will be described in the following paragraphs, there appears to be some functional partitioning between each type of repeat, but each type of repeat also has been found to have multiple affects. Most studies of repeats to date have focused on the affects of highly specific repeats within a particular context. No previous study has described the hierarchal nature of repeats identified by this thesis study, or the relationship of the hierarchal structures to protein function.

Direct repeats: identification and function

Most direct repeats are relatively short, often less than 60 nt, and consist of an identical or nearly identical sequence repeated from a few to several thousand times. When there are more than 5 tandem repeats, they are generally referred to as microsatellites. Direct repeats have been implicated in a large number of degenerative diseases and in many types of disease there is a dose-related affect between the number of repeats and the severity of the

disease. Some examples of these diseases are fragile X-associated tremor/ataxia syndrome (Leehey *et al*, 2007), Kennedy disease (Saunderson *et al*, 2007) and polyglutamine diseases (Shao *et al*, 2007).

Direct repeats are not necessarily contiguous. One of the earliest reports of direct repeats was in the Epstein-Barr virus (Kieff *et al*, 1982). It contains five tandem direct repeats that divide the genome into five functional domains. Some direct repeats increase gene expression - e.g. a hexanucleotide direct repeat in the erythropoletin enhancer is essential to the stimulation of red blood cell production (Blanchard *et al*, 1993). Direct repeats are a common component of hormone receptors, including thyroid hormone receptors (Desvergne, 1994), androgen receptors (Verrijdt, 2004) and vitamin D receptors (Shaffer and Gewirth, 2004). A direct repeat of the nuclear receptor hexameric DNA recognition motif (PuGGTCA) controls the expression of genes related to lipid metabolism (Schoonjans *et al*, 1996). Direct repeats appear to be an ancient mechanism by which transposable elements are integrated into genomes (She *et al*, 2002).

There are several websites and algorithms that can be used to analyze direct repeats. The algorithm by Gupta *et al* (2007) identifies exact and inexact tandem repeats. Domanic and Preparata (2007) provide an algorithm that does not require any a priori information about the size or type of pattern of the repeat. Sokol *et al* (2007) provide an algorithm in C++ that makes an exhaustive search for all tandem repeats in a sequence. Mrazek and Xie (2006) provide both a web service (http://www.cmbl.uga.edu/software.html) and source code to find direct and inverted repeats. RTAnalyzer (Lucier *et al*, 2007) is a web application that detects retrotransposons and L1 retrotransposition signatures (http://www.riboclub. org/cgi-bin/RTAnalyzer/index.pl?page=rt_find).

Complement match: identification and function

Although a literature search of Entrez http://www.ncbi.nlm.nih.gov/sites/entrez for "complement repeat" identifies six papers that contain this subject, all were found to describe repeats related to the immunological response protein complement. Complement repeats *do* occur in most sequences, and can be readily determined using the RePuter website (Kurtz *et al*, 2001) maintained by Bielefeld University, Germany (http://bibiserv.techfak.uni-bielefeld.de/reputer/submission.html). This type of repeat does

not appear to have been systematically evaluated, or to have been described as a result of its association with a notable functional aspect of a sequence.

Palindrome or inverted repeat: identification and function

Palindromes - also known as inverted repeats (IR) - are highly variable in length, from two to several thousand kb. As with all repeats, the identification of IRs is highly dependent on the symmetry criteria that are used. IRs are associated with multiple functional aspects of the genome, genes and RNAs, and for each of these genomic entities the function of the IR is related to its size.

Many proteins bind to DNA at short IRs - 4-20 bp. The most prominent class of these proteins is restriction enzymes. Modification sites for methylases and dimeric repressor proteins also bind short IRs (Sinden, 1994).

IRs may form cruciform structures (Platt, 1955; Gierer, 1966; Beerman and Lebowitz, 1973; Woodworth-Gutai and Lebowitz, 1976; Cox and Mirkin, 1997) in supercoiled, but not relaxed, DNA (Lilley, 1980; Panayotatos and Wells, 1981; Mizuuchi *et al*, 1982). IRs in DNA and RNA can fold back on itself, forming a hairpin structure (Sinden *et al*, 1994; White *et al*, 1972).

IRs occur near control regions of genes (Sinden, 1994) and are associated with a broad range of functional features. Palindromic motifs less than 25-bp are related to both expression and repression - e.g. the mercury resistance operon, mer, of the transposon Tn21 (Park *et al*, 1992). The T-antigen binding site within the core origin of replication of Simian Virus 40 is a perfect 24-bp palindrome consisting of a 12-bp primary unit - GAGGCC GAGGCG (itself an imperfect 6-bp repeat) and its 12-bp inverted repeat (Deb *et al*, 1986).

IRs occur at origins of DNA replication. In *E. coli,* these repeats are 254-bp. IRs occur at the origin of single-stranded bacteriophage genomes (in bacteriophage G4, 3 IRs, 20-44-bp each) and at the origin of eukaryotic viruses (in SV40, a perfect 27-bp IR, in herpes simplex virus, a 144-bp IR) (Sinden, 1994).

Short IRs adjacent to a double-stranded break (DSB) in DNA were found to promote the formation of large perfect DNA palindromes, which often lead to gene amplification (Tanaka *et al*, 2002). DNA damage - by radiation, hypoxia, chemicals and

replication errors - often results in DSBs (Lieber, 1998). The center of palindromic sequences of 71-bp, 230-bp and 225-bp were found to be the breakpoints that lead to the chromosomal rearrangements that result in der (22) syndrome, a severe congenital anomaly disorder in humans (Edelmann *et al*, 2001).

The RePuter web server (http://bibiserv.techfak.uni-bielefeld.de/reputer/submission.html) can be used to identify palindromes, and provide a visual display of their distribution. They can also be identified using the Mobyle server at the Pasteur Institute, France (http://mobyle.pasteur.fr/cgi-bin/MobylePortal/portal.py?form= palindrome). Programs to identify palindromes are components of the free EMBOSS software suite http://emboss.sourceforge.net/what/). Clustered regularly interspaced short palindromic repeats (CRISPRs) can also be identified on line (http://crispr.u-psud.fr/Server/CRISPRfinder.php) (Bland *et al*, 2007; Grissa *et al*, 2007).

To summarize, IRs are most commonly associated with gene regulation.

Mirror repeat or reverse match: identification and function

Mirror repeats have been noted in research studies when they are associated with a particular functional aspect of a gene or protein. For example, the bacterial transposon Tn7 recognizes and preferentially inserts into a mirror repeat which generates an intramolecular triplex when its complementary strand folds back onto the primary repeat unit (Rao *et al*, 2000); a mirror repeat is essential for the function of a *cis*-acting component of the Epstein-Barr virus OriLyt origin of replication (Portes-Sentis *et al*, 1997). IRs seem to be frequently (and incorrectly) identified as mirror repeats.

There have been few comprehensive studies of mirror repeats. Schroth and Ho (1995) evaluated the total number of inverted repeats (IR) and mirror repeats (MR) in segments of about 3 million base pairs for each of human, yeast and *E. coli* DNA sequences, and found all organisms were highly enriched in IRs, but only eukaryotes were enriched in MRs. They counted all occurrences of IRs and MRs and evaluated the potential of each to form cruciform structures. However, they investigated only perfect repeats - 100% identical except for a variable spacer of 3-6 nucleotides at the center of the repeat, and did not investigate the potential for hierarchal structure.

Cox and Mirkin (1997) re-examined IRs and MRs in seven species, which included eukaryotes, prokaryotes and archaebacteria. They found all species to be enriched in IRs and MRs, though differentially, and that enrichment was related to genome length. They also point out a highly significant factor that has not been systematically analyzed - "...a given sequence can often be considered as both a direct and a mirror repeat...This raises an important question: what fraction of our mirror repeats are also direct repeats?...(Cox and Mirkin, 1997, p. 5240). Like Schroth and Ho (1995), Cox and Mirken (1997) identified only perfect repeats. Although theses studies affirm the enrichment of mirror repeats in genomic DNA, they did not evaluate the relationship of repeats to sequence order, the impact of the hierarchal structure on the number of repeats, or the relationship of repeats to PSEs. The studies in this thesis, then, are the first systematic investigations of imperfect mirror repeats in an intragenic and genomic context.

Organization of repeats

The repeats that are described in the preceding pages - direct, complementary, palindrome and mirror - are identified based on the internal symmetry of the repeat unit. There are often multiple copies of individual repeats, organized in different ways throughout a genome.

Microsatellites are tandemly repeated sequence that arise by gene duplication (Singer and Berg, 1991). They are 2-6 nucleotides long, and may be repeated 5-5000 times. Microsatellites are found at many different loci in the genome in both coding and non-coding regions. Mono-, di- and tetra-nucleotide repeats are more common in non-coding DNA (Richard and Dujon, 1996; Field and Wills, 1998) and trinucleotide repeats (TNR) are most common in protein-coding DNA. TNRs result in tracts of simple sequence repeats (SSR) - repeats of the same amino acid.

In *Saccharomyces cerevisiae*, most proteins that contain SSRs are involved in transcription, signal transduction and cell growth and division (Young *et al*, 1999). The composition of TNRs is constrained and appears to be species specific. In yeast, AAA, AAG, AAC, AAT, AGC and ATC triplets predominate (Yount *et al*, 1999) and in primates, the only trinucleotide repeats show enrichment well above the threshold of statistical significance are CGG, CCG, CAG and/or GAA (Borstnik and Pumpernik, 2002). There is also a relationship between the length of the SSR and its composition. In

an analysis of all sequences in Genbank and Brookhaven Protein Data Bank (in 1994), glutamine SSRs (CAA, CAG) are most common in repeats greater than 10 amino acids, and leucine (CTN, TTA, TTG) most common in repeats less than 10 amino acids (Green and Wang, 1994).

The number of repeats within a microsatellite contributes to the polymorphism of a gene. This may affect a broad range of function in an organism, including social behavior (Hammock and Young, 2005). TNRs are also associated with a number of genetic diseases in humans (McMurray, 1995; Pearson and Sinden, 1996; Petruska *et al*, 1996; Paulson and Fischbeck, 1996; Sia *et al*, 1997a; Pandolfo, 1998).

Mobile genetic elements are genes or gene segments that can move throughout the genome and may be repeated, duplicated and relocated through mediation by either DNA or RNA. Transposons are DNA-based elements that can directly relocate within a replicon autonomously, via recombination; they occur in both prokaryotes and eukaryotes. Retroposons use RNA intermediates to relocate within a genome or within other hosts; they are found only in eukaryotes.

Retroposons include the gene reverse transcriptase (RTase), which is a component of retroviruses, long interspersed repetitive elements (LINES) or non-LTR-retrotrans-posons, processed retropseudogenes and short interspersed repetitive elements (SINES). "Retroviruses are thought to have evolved from LTR-retrotransposons by the acquisition of *env* genes, and LTR-retrotransposons in turn are believed to have evolved from LINEs by the acquisition of long terminal repeats (Eickbush, 1992 *in* Shedlock and Okada, 2000, p. 149). SINES are approximately 70-500-bp, and distributed throughout eukaryotic genomes in multiple copies (over 10^4), up to and sometimes comprising up to 13% of the genome (Lander, 2001); they do not encode enzymes essential for amplification, yet outnumber other repetitive elements. "Most SINES are derived from tRNA (Okada *et al*, 1995) and are believed to recombine and interact functionally with corresponding LINEs, leading to the acquisition of their retropositional activity (Okada *et al*, 1997 *in* Shedlock and Okada, 2000, p. 149). The Alu family of SINES is homologous to 7SL cytoplasmic RNA (Ullu and Tschudi, 1984). Alu repeats, present in 500,000 - 1 million copies in primates, contain an AGGTCA consensus sequence, which binds retinoic acid receptors (Vansant and Reynolds, 1995).

DNA motifs: identification

Most research related to DNA motifs is directed towards the identification of regulatory elements, especially the binding sites for transcription elements. Motifs can be identified by combining phylogenetic footprinting (the identification of orthologous sequences) (Blanchette and Tompa, 2002; Berezikov *et al*, 2004) with information about the identity of promoter sequences of co-regulated genes. Most of the programs that have been developed each work best for a specific organisms. Das and Dai (2007) provide an excellent review of these methods.

There are a number of web servers that can be used to identify regulatory motifs. These include: OSCAR (Jiang *et al*, 2007), ClusterDraw (Papatsenko, 2007) and rVISTA (Loots and Ovcharenko, 2004). Although DNA repeats are a form of motif, the identification of regulatory elements was not an objective of the studies comprising this thesis.

Protein motifs: identification

Protein motifs are the principal determinants of protein function. Although there are a core number of universal protein motifs, many are specific to particular levels of genome organization - species, proteins and/or cellular compartments. Therefore protein-based motif identification is greatly improved by matrix-based analysis rather than word-matching. Some of the web servers that can be used to identify protein motifs include ELM (http://elm.eu.org/#) (Puntervoll *et al*, 2003), MATLIGN (http://ekhidna.biocenter.helsinki.fi/poxo/matlign) (Kankainen and Löytynoja, 2007) and SMotif (http://caps.ncbs.res.in/SMotif/index.html) (Pugalenthi *et al*, 2007).

The relationship between rdIMRs and protein motifs was first identified by the systematic evaluation of IMRs in TnsA that was undertaken as a component of this thesis. In TnsA it was found that 88% of the potential protein motifs were translations of rdIMRs (Lang, 2005). In a subsequent study of the gag protein of HIV-1, it was found that the motifs most significant to the function of the protein were translations of the longest rdIMRs (Lang, 2007b). These studies together indicate that key functional motifs in a protein can be identified using the DNA from which it is translated. Furthermore, key motifs can be identified without knowledge of the tertiary structure of the protein.

2e. Protein secondary and tertiary structure: current methods of identification

The identification of the structure of a protein is a complex process. As with sequence comparisons, much of the information used by experimentalists is based on the similarity of a protein to those studied previously.

The amino acid sequence of a protein can be used to predict the secondary structure of the protein - the local composition of helices, turns and β–sheets, etc. Protein secondary structure nomenclature was first described by Kabsch and Sander (1983), in a document called the "Dictionary of Protein Structure," and is based on hydrogen bonding patterns first recognized by Pauling (1951). The PSEs recognized are: G = 3-turn helix; H = α-helix; I = pi helix; T = hydrogen bonded turn; E = β–sheet; B = residue in isolated β–bridge; S = bend. Information about helix- or sheet-forming propensities is combined with multiple sequence alignment, neighborhood information (other amino acids within ~7 residues), hydrophobicity and solvent accessibility, and used by neural networks, hidden Markov models and support vector machines to make predictions of secondary structure. The following models, which have achieved ~90% accuracy, are some of the currently available secondary structure prediction tools, which are currently available, on-line: **PsiPRED** (http://bioinf.cs.ucl.ac. uk/psipred/), **SAM-T02** (http://www.soe.ucsc.edu/research/ compbio/HMM-apps/T02-query.html), **PORTER** (http://distill.ucd.ie/porter/), **SABLE** (http://sable.cchmc.org/).

The secondary structure of a protein can be used to infer its tertiary structure, by comparing its secondary structure to those of proteins for which the tertiary configuration has been determined experimentally. This process is called protein threading or structural alignment. There are an extensive number of websites devoted to this project, including **Combinatorial Extension -- Monte Carlo** (http://cl.sdsc.edu/), **DaliLite** (http://www.ebi.ac.uk/ DaliLite/), **VAST** (http://www. ncbi.nlm.nih.gov/Structure/VAST/vastsearch.html), **phyre** (http://www.sbg.bio.ic.ac.uk/ ~phyre/) and **Swiss-Model** (http://swissmodel.expasy.org/ SWISS-MODEL.html).

Current methods of the computational prediction of protein tertiary structure are also based on protein sequence homology - the amino acid sequence of the unknown protein is compared, by pairwise alignment, to the amino acid sequence of proteins of

experimentally determined structures. Crystallographers, using processes and methods that are necessarily complex and time-consuming, perform the experimental determination of tertiary structures. Computational comparison with known structures can be made at several web sites that provide these services, including **phyre** at the Imperial College of London (http://www.sbg.bio.ic.ac.uk/~phyre/) and **Swiss-Model** at the Swiss Institute of Bioinformatics (http://swissmodel.expasy.org/ SWISS-MODEL.html). Computational predictions always contain disclaimers as both sequence and environmental variables may have major, previously undocumented affects on the predicted protein conformation, and all predictions require various forms of experimental verification.

The studies contained in this thesis provide a relatively simple method for identifying key functional sites in a protein. The application of this information to predicted tertiary structures provides a more informed estimate of the significance of various sites in the protein, and improved prediction of the possible affects of sequence differences on tertiary structure.

Some segments of proteins consist of amino acid repeats. The same types of potential repeats exist as in DNA - direct, complementary, inverted and mirror - except that all are single stranded. Most of the amino acid repeats that have been described are proposed to have arisen by gene duplication (Heringa, 1998). There are several computer programs that identify protein repeats - including TRUST (Szklarczyk and Heringa, 2004) and AuberGene (Szklarczyk and Heringa, 2006). The distribution of amino acid repeats has not been evaluated in the studies comprising this thesis.

2f. Summary of novel methods developed in this thesis

The methods described in this thesis are based on long-established principles of sequence order and dinucleotide composition, but are an expansion and integration of them. They include new methods for the analysis of dinucleotide patterns and dinucleotide composition, and evaluate previously unexamined relationships between DNA and protein structure and function. Several hypotheses are evaluated. Do imperfect mirror repeats coincide with protein structural elements? Does net dinucleotide composition provide the same species-specific identity as dinucleotide composition? Can DNA be used to identify

key functional and/or structural components of proteins? The research encompassed by this thesis demonstrates that the answers to these questions are affirmative.

Historically and currently, identification and characterization of genomes, genes and gene functions are based on comparisons of DNA sequences that are mostly imperfect matches, often with an identity as low as 50%. In contrast, previous analyses of DNA repeats have selected for those repeats that are most perfect, but almost always with an identity ≥80%. Perhaps the most significant new methodolgy described in this thesis and absent from previous studies of repeats, is the evidence for the sequential and hierarchal ordering of repeats throughout a genome, and the relationship of these repeats to significant and stable functional motifs in the translated proteins. This thesis demonstrates these principles by the systematic evaluation of *imperfect* mirror repeats that have relatively low values of symmetry (≥ 50%) in a broad range of genomic and protein contexts. Lang (2005) evaluates the relationship between the distribution of IMRs and PSEs in 17 diverse genes. Lang (2007b) similarly evaluates this distribution in the gag polyprotein of HIV-1, and additionally the relationship between IMR length and functional significance in the translated protein and the relationship between IMRs and cleavage sites in a polyprotein.

This thesis also investigates whether the long-established principle that organisms have a characteristic dinucleotide composition can be expanded to include "net dinucleotide" identity - i.e. sXY + tYX = (s-t)XY. Lang (2000) describes the method and evaluates its output in tRNAs, ribosomes and several proteins. Lang (2007a) evaluates a highly dissimilar set of HIV genomes in order to determine whether net dinucleotide values might provide a unique identity related to gene function.

Each of these studies (Lang 2000, 2005, 2007a, 2007b) provide evidence that challenges a long-held hypothesis - that DNA is a template subject to random mutation and subsequent selection that is otherwise independent of its translated protein. A brief description the new methods and analyses constituting this thesis is provided within the following pages and a detailed description is contained within the reprints of published papers included as addendum to this thesis.

3. Circuit assemblages, derived from net dinucleotide composition characterize RNAs, genes and gene function within a species

As described previously, the traditional basis of genome and gene identification is pairwise comparative analysis, which compares the sequence composition at each sequence position between organisms or genes, to arrive at a quantitative measure of similarity. In contrast, dinucleotide composition is a non-linear, yet inherent measure of sequence composition and position which it is known to distinguish organisms and genes from each other. This thesis tests an expansion of the well-established properties of dinucleotide composition, by investigating the hypothesis that "net dinucleotides" - i.e. the dominant dinucleotides of a sequence - characterize and distinguish organisms and genes.

3a. The method of calculating net dinucleotides

Net dinucleotides can be calculated for any type or length of sequence, but the selection of functional increments results in more informative results. The functional increments may be whole genomes, operons, genes, or PSEs. **Figure 2** illustrates the transformation of dinucleotides into net dinucleotides.

```
acccaccaaggcaaagagaagagtggtgcagagaga  36 nt
.......................tg.tg........  2tg
.......................gt.gt........  2gt
...............gaga.ga........gagaga  6ga
..........gc...............gc.......  2gc
.........gg.............gg..........  2gg
........ag....agag.agag......agagag.  8ag
.......aa...aaa...aa................  4aa
...ca.ca...ca...............ca......  4ca
.ccc.cc.............................  3cc
ac..ac..............................  2ac

            2gt + 2tg =   0
            0cg + 2gc =  2nGC
            8ag + 6ga =  2nAG
            2ac + 4ca =  2nCA
```

Figure 2. The composition of any DNA sequence can be represented as a dinucleotide count or net dinucleotide count. The value of a net dinucleotide measure is based on the assumption that each dinucleotide represents some value that is equal in quantity but opposite in sign from its reversed dinucleotide, i.e. AC = -CA, AG = -GA, AT = -TA, CG = -GC, CT = -TC, GT = -TG.

Net dinucleotides of equal value can be clustered, providing a shorthand for gene sequences of any length. For example, using the values of net dinucleotides from Figure 2, 2nGC, 2nAG, 2nCA can be combined into the circuit 2nGCAG. All the circuits that are derived from a sequence are collectively called its circuit assemblage (CA).

The CAs of a sequence describes the dominant dinucleotides of a sequence that occur in equal numbers. CAs provide a succint notation that has virtually unlimited quantitative and qualitative variability, despite an apparent simplicity. There are 14 possible types or circuits: GCAG, GACG, GATG, GTAG, GCTG, GTCG, ATCA, ACTA, GCTAG, GATCG, GTCAG, GACTG, GCATG, GTACG. These can be combined in 5040 (7!) different ways as reverse circuits cannot coexist within the same sequence. The value of each circuit can vary within each combination.

3b. DNA walks are graphical representations of net dinucleotide circuits

Graphical methods of surveying sequence variability have provided several significant insights into functional aspects of genome composition. Mizraji and Ninio (1985) substituted symbols for G, C, A and T and found that coding and non-coding region of the hemoglobin gene could readily be distinguished on this basis; they also proposed a method of vectorial coding that demonstrated that coding regions remained parallel to a fixed direction, while the trajectories of introns were inhomogeneous. Lobry (1996) modified the vectorial pattern used by Mizraji and Ninio (1985) and found that both the origin of replication and terminus of replication of bacteria could readily be detected based on the acute change of trajectory of the vectorial representation of the sequence of the genome. An alternate method proposed by Cebrat and Dudek (1998) made it possible to readily distinguish between coding and non-coding sequences and to identify the coding strand. One of the earliest DNA walks for a eukaryote was performed by Abramson *et al* (1998) - for *Saccharomyces cerevisiae*. DNA walks are now available for an extensive number of organisms at http://www2.unil.ch/comparativegenometrics/ (Roten *et al*, 2002). GraphDNA is a recently developed program that conveniently performs a DNA walk and analyzes multiple types of compositional skews (Thomas *et al*, 2007).

A DNA walk is a graphical representation (on an xy axis) of the DNA sequence that is created by assigning each nucleotide a particular direction. In the GraphDNA program

(Thomas *et al*, 2007) the assignments are: A = -x, T = +x, G = -y, C = +y. By this method each dinucleotide becomes a vector within the graph. DNA walks have not previously been qualitatively related to composition, but the recognition of net dinucleotide circuits and their functional associations now makes this an appealing prospect for future study.

The trajectory of the graphed sequence (DNA walk) reflects net dinucelotide values. Net dinucleotide circuits occur where those segments of the graph overlap, intersect and/or cluster together. **Figure 3** illustrates a DNA walk for human preproglucagon (Accession:V01515) and the position of its four exons.

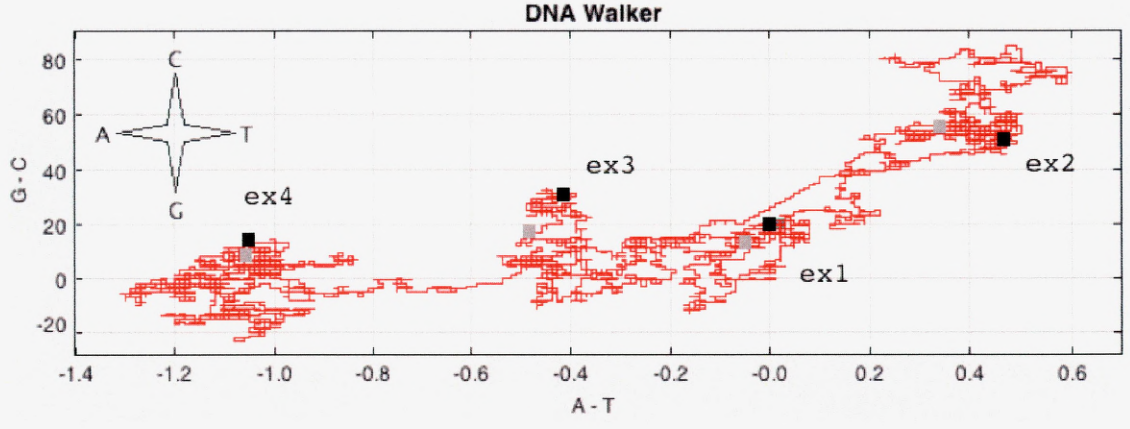
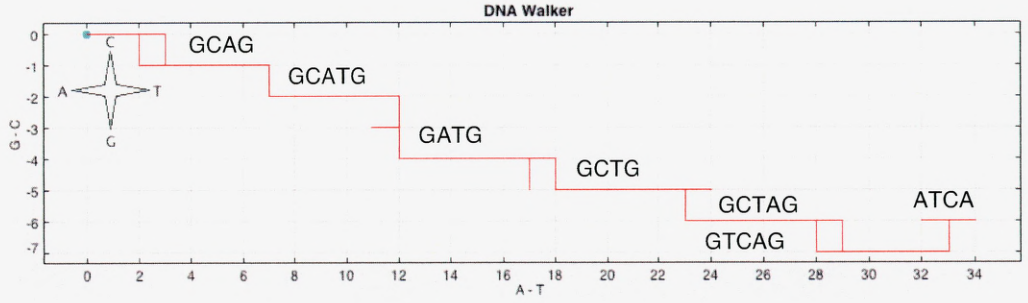


Figure 3. A DNA walk for human preproglucagon (Accession:V01515) and the position of its four exons. The start of the exon is indicated by a black square, and the end by a grey square. Ex1 is the signal peptide. Ex2 is glucagon. Exons 3 and 4 are GLP-1 and GLP-2, two peptides homologous to glucagon but not identical. It can be observed that each of these exons is confined to a distinct region of the entire DNA walk, and each is part of a distinctive cluster.

The geometric basis of the DNA walk inherently results in a tendency for the trajectory of the sequence to fold back upon itself when a sequence makes a circuit. When circuits are abundant - as they are in protein coding DNA (Lang, 2005, 2007a) - the trajectory has the appearance of an irregular cluster (Figure 3). This feature is illustrated in more detail in **Figure 4.** The lack of a unique vector value for each circuit suggests that the current format (4-fold movement) does not adequately elucidate the variability of the sequence. More explicit formats, such as 3D mapping, or other angles for vectors, are interesting possibilities for future study.

```
>circuits
TTTGCAGTTTTTGCATGTTTTTGATGTTTTTGCTGTTTTTGCTAGTTTTTGTCAGTTTTTATCATTT
            GCAG        ( 3,-1)..( 2,-1)  =   (-1, 0)
            GCATG       ( 7,-2)..( 7,-2)  =   ( 0, 0)
            GATG        (12,-3)..(12,-4)  =   ( 0,-1)
            GCTG        (17,-5)..(18,-5)  =   ( 1, 0)
            GCTAG       (23,-6)..(23,-6)  =   ( 0, 0)
            GTCAG       (28,-7)..(28,-7)  =   ( 0, 0)
            ATCA        (33,-7)..(32,-6)  =   (-1, 1)
```




```
>circuits-reversed
TTTGACGTTTTTGTACGTTTTTGTAGTTTTTGTCGTTTTTGATCGTTTTTGACTGTTTTTACTATTT
            GACG        ( 3,-1)..( 2,-1)  =   (-1, 0)
            GTACG       ( 7,-2)..( 7,-2)  =   ( 0, 0)
            GTAG        (12,-3)..(12,-4)  =   ( 0,-1)
            GTCG        (17,-5)..(18,-5)  =   ( 1, 0)
            GATCG       (23,-6)..(23,-6)  =   ( 0, 0)
            GACTG       (28,-7)..(28,-7)  =   ( 0, 0)
            ACTA        (32,-7)..(32,-6)  =   ( 0, 1)
```

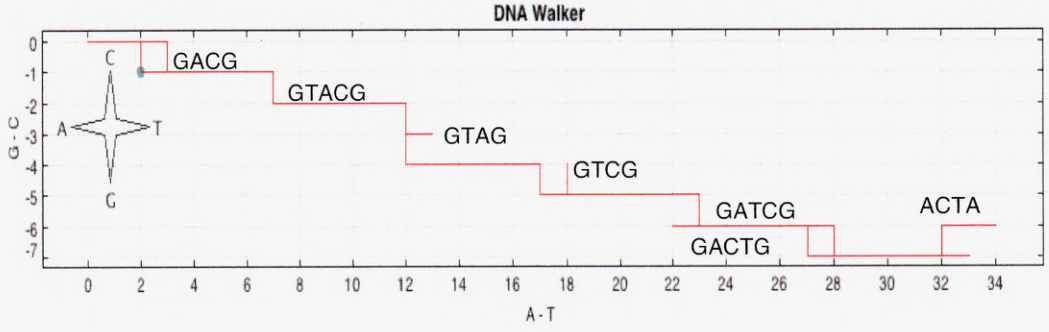
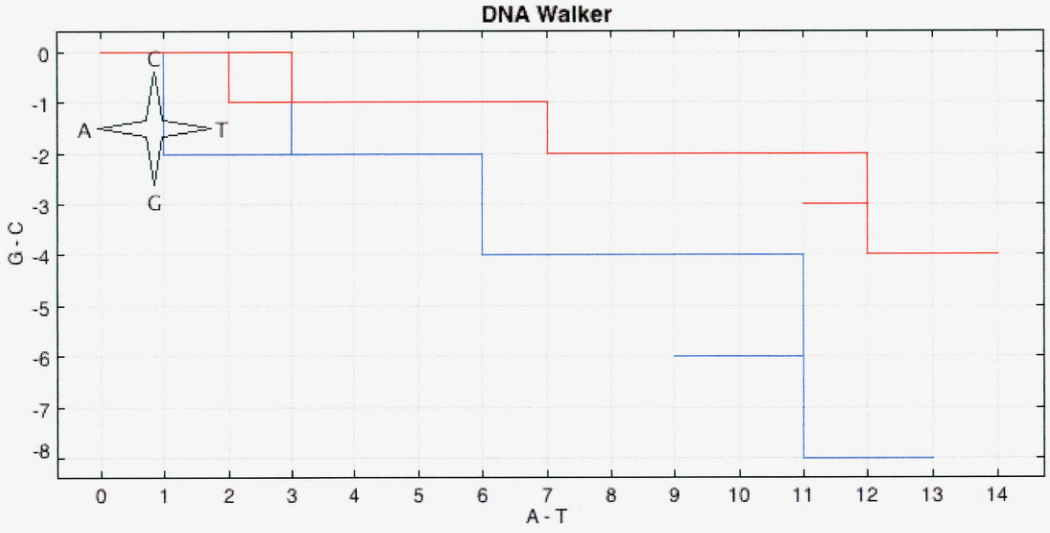


Figure 4. Overlaps and clusters in DNA walks can be qualitatively described as net dinucleotide circuits. Sequences used for each of these graphs are shown above the graph. All possible circuits are illustrated by constructing sequences that contain a circuit (value 1) separated by "TTTTT." GraphDNA (Thomas *et al*, 2007) was used to produce these diagrams.

circuits		co-ordinate change
GCAG	GACG	(-1, 0)
GATG	GTAG	(0,-1)
ATCA		(-1, 1)
GCTG	GTCG	(1, 0)
ACTA		(0, 1)
GCATG	GTACG	(0, 0)
GCTAG	GATCG	(0, 0)
GTCAG	GACTG	(0, 0)

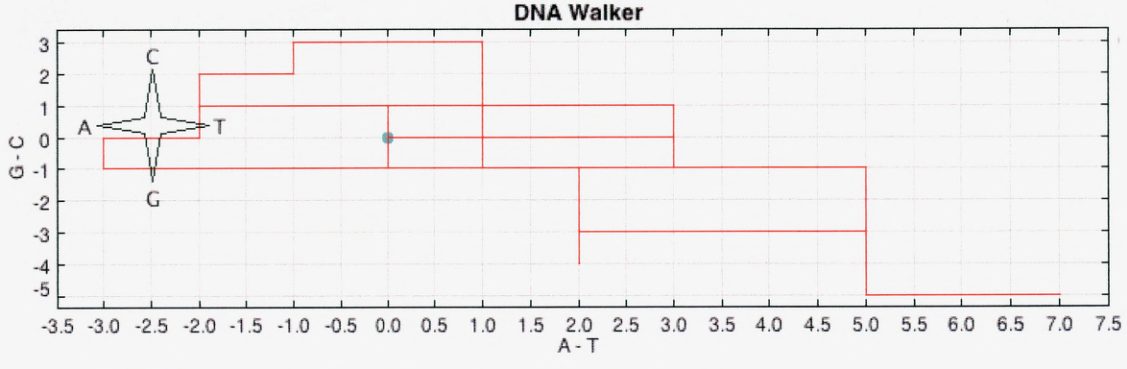
Figure 5. Using the GraphDNA format for the DNA walk, some circuits have a unique impact on the movement of the trajectory and others have the same impact. Further investigation of other formats, such as 45° vectors described in Mizraji and Ninio (1985) might result in unique values for each circuit. This may make it possible to directly relate findings based on the DNA walk to protein structure.

A DNA walk inherently creates net dinucleotide circuits because reverse dinucleotides cause retracing of a former path. In a graph, the retracing may not take place in the exact position, but the net trajectory will be the equivalent of "net dinucleotides." **Figure 6** illustrates the trajectory of a few circuits when represented by a single nucleotide and homodinucleotides. In **Figure 7**, the sequences of Figures 6 are embellished with additional nucleotides of a composition and position that creates net dinucleotide circuits of the same composition. The trace of this sequence is noisier, with multiple retraced positions, but the terminus has a direction that is similar to the more simple data.




```
>GCAG-GCATG-GATG  (14,-3)
tttGCAGttttGCATGttttGATGttt
>GCAG-GCATG-GATG-homo-dinucleotides (13, -8)
tttGGCCAAGGttttGGCCAATTGGttttGGAATTGGttt
```

Figure 6. Doublets within circuits increase the length of a trajectory but maintain the same shape. The DNA walk of each sequence is shown in the color of its corresponding line on the graph. The net trajectory of each sequence is shown in parenthesis.




```
>GCAG-GCATG-GATG-net-dinucleotides  (7,-5)
tttGCCGCAACCAAGAGGAGtttttGGGCCGCCACCAAATTGGttttGGAAATTATTGGttt
```

Figure 7. Each circuit segment within the sequence illustrated in red in Figure 6 is padded with extra dinucleotides arranged so that they evaluate to the same net dinucleotide composition as in Figure 6. This padded sequence more closely approximates the noisy distribution typical of protein-coding regions. The trend of the trajectory is maintained although there are significantly more back folding pathways.

3c. Circuit composition reflects the type of sequence repeat

The circuit assemblage of any sequence is related to the type of repeat within the sequence. This feature is illustrated in **Figure 8** and summarized in **Table 1,** using the first half of the TAR element of HIV-1, which is known to form a hairpin in its entirety that is highly significant to strand transfer (Heilman-Miller *et al*, 2004).

type of repeat	length	GG	CC	TT	AA	GTCAG	GCTAG	GCAG	GTAG	TAG
unit	24	2	3	2	0			1	2	GC
A. complement	48	4	4	2	3	3				no
B. palindrome	48	4	3	2	3		3			GTAC
C. forward-direct-tandem	48	4	4	4	1			1	4	GC
D. mirror-repeat	48	4	5	4	1			1		no

Table 1. The composition of a circuit over the interval of a repeat is based on the type of repeat. This table summarizes the circuit values of the sequences illustrated in Figure 3, which are different types of repeats generated from the same repeat seed - the first half of the TAR element of HIV-1.

Figure 8. The first half of the HIV-1 TAR element (Heilman-Miller *et al*, 2004), which is known to form a hairpin structure, is used as the unit of repeat (shaded in turquoise), to illustrate the circuit assemblage differences of each type of repeat. Two hypothetical modifications were made to the repeated component in order to make them "imperfect." The exact repeat is indicated above the main line of the sequence; the modified nucleotides are indicated in yellow. In each type of repeat it was possible to convert one sequence of ccc to ctc. In A and B, one sequence of gga was converted to gaa; in C, agc was converted to aac; in D, cga was converted to caa.

3d. Circuit composition provides a unique identity for RNAs and genes within an organism

The concept of net dinucleotide count was first described and defined by Lang (2000). This paper demonstrates that net dinucleotide analysis may provide a meaningful shorthand for RNA and gene identity and its potential use in multiple aspects of genome analysis, including the analysis of introns and exons, and biochemical pathways. The paper summarizes the modification of CAs accompanying the excision of insulin from preproinsulin. It demonstrates that CAs identify the same types of groups as traditional phylogenetic analysis, using the example of *Monilinia* rRNAs and demonstrates that the tRNAs of a single species are uniquely defined by their circuit composition.

In subsequent unpublished studies, Lang verified the within-organism uniqueness of tRNA composition for numerous other organisms. CAs for sea urchin tRNAs are summarized below in **Table 2**. This table also demonstrates that each class of aminoacyl-tRNA synthetase has characteristic circuits. This specificity suggests two other applications for CAs. First, the CAs of tRNA may provide a quantitative basis for the definition of a species. Next, the CAs of tRNAs may serve as eigenvalues for all genes or gene pathways within an organism.

Sea Urchin Genome	GTCG	GATCG	ATCA	GTCAG	GTACG	GTAG	GCTAG	GCAG	GCATG	GACG	GATG	GACTG	ACTA	GCTG	TAG
ProTGG.830.Scaff49098.trna3.(24022-23951).72bp.64.									1						GATC
AlaGGC.Scaff1039.trna1.(159822-159895).74bp.29.61			1					3							GC
ThrAGT.1997.Scaff104trna4.(83115-83187).73bp.65.59						1	1	2							GT
AsnGTT.Scaff432.trna10.(171352-171425).74bp.75.63		1					1								no
SerGCT.1965.Scaff140575.trna1.(430-511).82bp.85.34		2									1				no
AspGTC.Scaff100638.trna1.(23665-23736).197.72bp.7	2					1									TCG
HisGTG.445.Scaff109571.trna2.(27105-27034).72bp.69	1		1												GTCA
PheAAA.869.Scaff119087.trna1.(530-465).66bp.49.37	3				1					1					GA
GlyGCC.369.Scaff78076.trna4.(2944-3014).71bp.73.37		2	1	1											GTCA
LysCTT.1480.Scaff120291.trna1.(25788-25714).75bp.2				3											no
CysGCA.240.Scaff74307.trna2.(48588-48517).72bp.63				2											GCAT
MetCAT.975.Scaff323.trna1.(5820-5892).73bp.55.06	2		1												GTCA
TyrGTA.2082.Scaff54298.trna13.(10679-10767).89bp.7	2														CGTA
TrpCCA.2063.Scaff60939.trna5.(573-500).74bp.62.9	3	1													GTCA
GluCTC.303.Scaff1083.trna2.(104585-104656).72bp.69	2	1													TCA
LeuTAA.605.Scaff40491.trna2.(10747-10829).83bp.71.	2				2										no
IleAAT.471.Scaff31137.trna11.(15088-15016).73bp.68.	3														GTCA
ValAAC.2107.Scaff88681.trna9.(33074-33002).73bp.76	3				1	2									GT
GlnTTG.283.Scaff1148.trna2.(179895-179966).72bp.62			1												GAT
ArgACG.58.Scaff74211.trna7.(3955-4027).73bp.60.92			2						1						no

Class II aminoacyl-tRNA synthetase Class I aminoacyl-tRNA synthetase

Table 2. In the sea urchin and more than 20 other species evaluated (unpublished), each tRNA species within an organism is unique. This suggests CAs may provide an informative shorthand applicable to regulatory pathways and gene families within each genome. This hypothesis is further supported by the finding that the phylogenetic groups distinguished by CAs are the same as those determined by traditional phylogenetic analysis (Lang, 2000). The tRNA sequences in this table were obtained from the tRNA database at UCSC, USA (http://lowelab.ucsc.edu/GtRNAdb/Spurp/Spurp-tRNAs.fa).

3e. A database of RNAs and genes from different organisms indicates that some circuits appear to be associated with particular gene functions, but others are species-specific

Most of the sequences analyzed in Lang (2005) were tRNAs and rRNAs. The finding that each tRNA within a species had a unique CA suggested that CAs might also distinguish proteins by function. To evaluate this hypothesis, a database was created that included tRNAs, rRNAs and more than 2000 genes. The genes were selected randomly from recent publications, using the guideline that a diverse group of genes should be included, in order to maximize the predictive potential of the database. The tRNAs were selected from the genomic tRNA database at UC Santa Cruz (http://lowelab.ucsc.edu/ GtRNAdb/). The following fields are included in the database: accession number, date, gene name, collection, tissue, organism, notes, product/function, start and stop, complete/incomplete,

CDS or intron/exon information, length, nucleotide, dinucleotide, net dinucleotide and circuit composition. At the present time there are more than 2000 genes in the database.

Examples from the database are provided in the following paragraphs, **Tables 3 and 4, and Figures 9 and 10.** These tables and figures all demonstrate that genes from different species that have a similar function frequently have one or two circuits that are similar. However, in most cases, there are species-related differences in circuits. After summarizing the distribution of circuits, e.g. Tables 3 and 4, it was determined that the database is too small to be used for statistically significant evaluations of the association between circuit type and function, and that the required size was not attainable by a singular effort. The existing database, however, provides sufficient evidence that the hypothesis, reframed, could be tested. This was accomplished by making a comprehensive analysis of the genes of a single species - HIV (Lang, 2007a). The basis for the decision to comprehensively analyze a single species is illustrated in the following paragraphs, tables and figures.

Tables 3 and 4 illustrate some of the types of information that can be extracted from the database. Multiple fields can be selected at one time, to collect specific CAs. Any field can be searched - including nucleotides and dinucleotides.

Some examples of proteins with high values for each type of circuit are described in the following paragraphs. These are summarized using tetrahedral diagrams in **Figures 9 and 10**. The circuit GCAG is most common in the HIV-1 genome. There are >163 GCAG circuits in all HIV-1 genomes. Genes (in this database) that have the most GCAG after HIV are the mouse clock gene (AF000998, 81GCAG), the SARS genome (AY278741, 72GCAG), mouse talin-C (X56123, 62GCAG) and leptin (U50365, 42GCAG). Most
common in the reverse circuit - GACG - are *E. coli* synthetase (X53864, 16GACG), *E. coli* ferrichrome (U70214, 8GACG) and A. vinelandi flavodoxin (M20568, 7GACG).

The circuit GATG is most common in insulin receptors (*C. elegans* AF012437, 115GATG; human, X02160, 69GATG) and ferrochrome non-transport proteins (*E. coli*, U70214, 86GATG; *Strongylocentrotus purpuratus*, L34680, 77GATG). The reverse circuit - GTAG - is most common in the complete mitochondrial genome of *C. elegans*

(X54252, 101GTAG) and other genes within it, such as ND5 (33GATG) and ND6 (16GATG).

organism	Accession	Gene Name	total bp	ACTA	ATCA	GCAG	GACG	GCTG	GTCG	GATG	GTAG	GCATG	GCTAG	GACTG	GTACG
Rickettsia (Andersson)	U11018	23SrRNA	**2699**	16							5		1		
E. coli	U70214	23SrRNA	**2904**	16										16	
Thermus thermophilus	M74795	methylase	**1287**	12				4						6	
Thermus thermophilus	M76682	methylase - M.TthHBB1	**1287**	12				4						6	
human	J01415	mRNA 7	**1668**	11				16					19		
C. elegans	X54252	Co I protein	**1578**	10							5		14		
human	J01415	NADH, subunit 3	**346**	9				3							
human	J01415	NADH dehydrogenase subunit 4	**1378**	9				13					11		
Rickettsia prowazekii	M21789	16SrRNA	**1508**	9				4				2			
E. coli	U70214	ferrochrome non-receptor protein	**1**	8			8								3
human	J01415	deletion > ophthalmoplegia	**3843**	7							2		64		
human	J01415	deleted > syndrome	**4190**	7				14							
Rickettsia prowazekii	Z54170	tuf, elongation factor EF-Tu CDS	**1185**	7				14					11		
human	J01415	mRNA-13	**958**	6				13						1	
human	J01415	mRNA 14	**843**	6								14	2		
A. thaliana	M91208	LEAFY, alt ex3	**582**	6							2				
human	J01415	RNA13	**958**	6				13						1	
human	J04809	adenylate kinase - AK1 - exon 5	**117**	6				1		1		3			
A. thaliana	M91208	LEAFY, alt exon3	**558**	5							3				
A. thaliana	M91208	LEAFY, alt exon3	**524**	5							2				
A. thaliana	M91208	LEAFY, alt ex3	**519**	5							2				
SARS	AY278741	X2 protein	**465**	5				2						3	

Table 3. The circuit ACTA is most common in 23S rRNA and methylase. However, other circuits are specific to each organism. The 23SrRNA gene in both *E. coli* and Rickettsia have 16ACTA, but Rickettsia also contains 5GTAG and *E. coli* also contains 16GACTG.

organism	Accession	Gene Name	total bp	ATCA	ACTA	GCAG	GACG	GCTG	GTCG	GATG	GTAG	GCATG
Caenorhabditis elegans	L29052	Cell death protein (ced-3) all introns	3432	64						46		
C. elegans	L46861	talin - C set	2046	54						28		7
C. elegans	AF005205	daf-3	2391	48						41		
D.yakuba	X61127	period gene, CDS	1	43		41						20
C. elegans	L46861	talin - D set	2361	43						34		2
Drosophila	X96926	eggshell	930	43								42
C. elegans	L46861	talin - B set	1767	42						18		3
human	J02933	A1 wo optional intron	3642	40		32			11			
human	J02933	A1 with optional intron	3709	40		33						11
human	J02933	blood coagulation factor VII	1	33		9						58
P. chrysosporium	Z24723	ubiquitin CDS	1146	28					17			
Strongylocentrotus purpuratus	L34680	calcium-binding protein CDS	4683	27						77		26
A. thaliana Wassilewskija	U03456	Luminidependens	2862	26		12						48
C. elegans	U72883	daf-7	1053	25						20		
Transposon Tn1546	M97297	transposase	2967	24						2		36
C. elegans	L23110	daf-4	2235	22						57		
A. thaliana	X53579	agamous (AG)	855	20						1		1
human	J02933	blood coagulation factor VII, intronC	1929	20		43						
Caenorhabditis elegans	L29052	cell death protein (ced-3) intron 3-4	1195	20						28		
A. thaliana Wassilewskija	U03456	Luminidependens, number13	570	19		9						2
Caenorhabditis elegans	L29052	cell death protein (ced-3) intron 4-5	912	17						8		
Caenorhabditis elegans	L29052	Cell death protein (ced-3) CDS	1512	17						27		
Dictostelium discoideum	U03413	calcium binding protein (AX2) CDS	1404	16		41						
Dictyostelium discoideum	U03413	calcium binding protein (AX2) CDS	1233	15		42						
A. thaliana	U12546	APETALA2	1299	14						11		
Stylonychia mytilis	X61748	telomere-binding protein, B subunit	1179	14		15						

Table 4. The reverse circuit ATCA is most common in genes for cell death, talin and eggshell and Ca-binding protein. The Ca-binding protein of both *C. purpuratus* and *D. discoideum* contain high amounts of ATCA, but *C. purpuratus* has 77GATG and *D. discoideum* has 41GCAG.

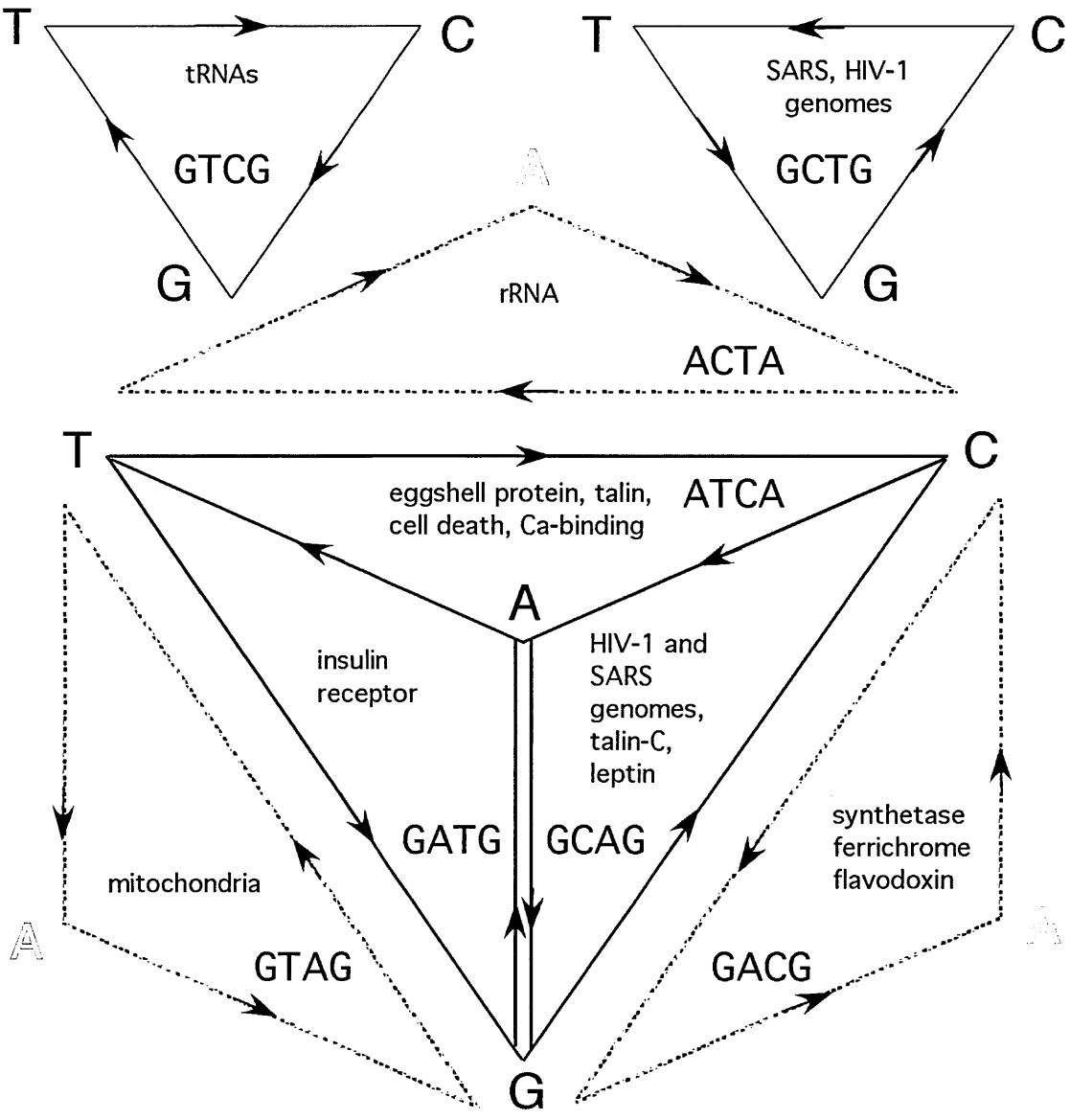


Figure 9. Each 4-mer circuit can be represented as a face of a tetrahedron. The direction of each circuit is represented by arrows; the reverse circuit of each tetrahedral face is opposite it. Proteins having the highest numbers of a particular circuit - within the database designed for this study - are indicated on each face, and its reverse. For example, GCAG circuits are highest in the HIV-1 and SARS genomes, talin-C and leptin. GACG circuits are highest in synthetase, ferrichrome and flavodoxin. Accession numbers and more detailed descriptions can be found in the text.

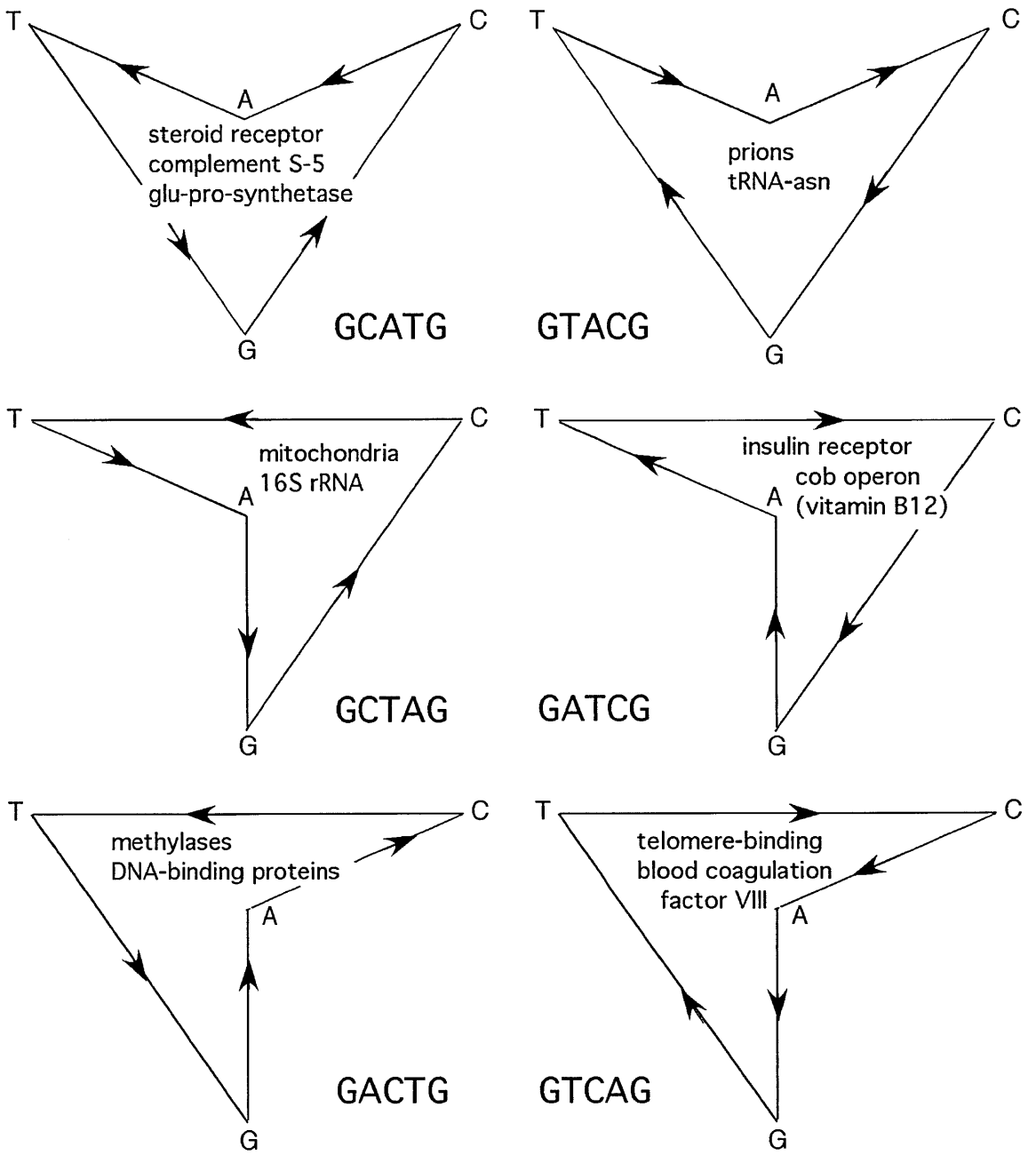


Figure 10. Tetrahedral representations of 5-mer circuits. Genes with the highest number of each circuit (in this database) are indicated. Accession numbers and additional proteins for each circuit are described in the text.

The circuit GCATG is most common in human ioduronate 2-sulfatase (AF011889, 172GCATG), a sulfatase enzyme associated with Hunter syndrome. High GCATG values also occur in SARS (AY278741, 154GCATG), the human A1B1 steroid receptor coactivator (AF012108, 131GCATG), human complement component C5 (M57729, 105GCATG) and glutamyl-prolyl-tRNA synthetase (EPRS) (NM_004446, 80GCATG). The reverse circuit - GTACG - was not found in high numbers, but was most common in prions (S68626, 5GTACG) and SARS LTR (AY278741, 3GTACG) and tRNAs (X52791, tRNA-asnU, 2GTACG; and others).

The circuit GCTAG is far more common. It is most common circuit of the human mitochria genome (J01415, 142GCTAG) and also high in the *C. elegans* mitochondrial genome (X54252, 78GCTAG). Lesser numbers predominate in hum 16SrRNA (J01415, 24GCTAG) and 12S rRNA ((J01415, 19GCTAG).

GATCG is most common in daf-2, the *C. elegans* insulin receptor (AF012437, 120GATCG). It is also common in the genes of the cob operon (J..N) of *Pseudomonas denitrificans* essential to the synthesis of vitamin B12 (M62869, 13-39GATCG).

GTCAG is most common in intron A of human blood coagulation factor VII (J02933, 33GTCAG), Ca-binding proteins in *Dictostelium discoideum* (U03413, 21GTCAG), telomere-binding protein in *Stylonychia mytilis* (X61748, 12GTCAG) and a seed storage protein (CRA1) of *A. thaliana* (X14312, 11 GTCAG).

GACTG is most common in *E. coli* 23SrRNA (U70214, 16GACTG), human myogenic factor 3 (MYOD1) (NM_002478, 6GACTG), and methylases and DNA-binding proteins in several species (3-6GACTG).

Although, as mentioned previously, a much larger database will be needed to fully evaluate the relationship between circuits and gene function, the database constructed as part of this thesis provides clear evidence that orthologous genes have circuits of similar composition and value.

3f. Circuit assemblages distinguish genes within a single organism, despite significant sequence variability

Dinucleotide composition and CAs were determined and analyzed for functional components at multiple levels of organization for a highly diverse group of 110 HIV

genomes. The data set for this study included 110 genomes (from 14 subtypes) of extremely heterogeneous HIV. The sequence identity between subtypes differed by as much as 30%. The results of the study provide a new perspective of genome organization and evidence of a mechanism that operates to conserve species identity despite mutation and evolution. This research is summarized in the attached publication "Circuit assemblages derived from net dinucleotide values provide a succinct identity for the HIV-1 genome and each of its genes" (Lang, 2007a). Some of the key results are listed below.

i. The circuit assemblage of each genome (~10,000 nt) differs by less than 18 net dinucleotides despite up to 30% sequence variability.
ii. Each gene of HIV-1 has a unique circuit assemblage that is conserved within all subtypes.
iii. Unique circuit assemblages distinguish each mature protein of the *gag* polyprotein, and each functional component of the hypervariable *env* gene.
iv. Circuit assemblages point to sequence differences between HIV groups that have remained localized and those that have become pandemic, a feature that may be applicable to the evolution of pathogenicity and antibiotic resistance.

4. Imperfect DNA mirror repeats (IMRs) coincide with protein structural elements (PSEs) and identify key functional sites in the translated protein

A DNA mirror repeat is a sequence segment delimited on the basis of its containing a center of symmetry on a single strand and identical terminal nucleotides, e.g.

```
--------> <--------
5'- G C A T G G T A C G - 3'.
```

IMRs often contain non-symmetrical elements, the proportion of which can be described by the percent of symmetry. For example, in the segment

```
--------> <--------
5'- G C T T G G T A C G - 3'
```

eight of ten nucleotides, or 80% are symmetrical. One of the major goals of the thesis was to to evaluate the relationship between DNA mirror repeats and protein motifs, a decision based on the observation that many of the key functional sites in proteins appeared to be

the translation of IMRs. Preliminary evaluation of this feature indicated that the frequency of mirror repeats was high, and that a comprehensive systematic approach would be required. To address this issue, several perl programs were written (Appendix, Part B), then used to analyze a broad variety of proteins.

The paper - *Imperfect DNA mirror repeats in E. coli TnsA and other protein-coding DNA* (Lang, 2005) - defines two types of mirror repeats, then examines the relationship of each to PSEs in 17 proteins, mostly human, from various cellular locations. The paper - *Imperfect DNA mirror repeats in the gag gene of HIV-1 (HXB2) identify key functional domains and coincide with protein structural elements in each of the mature proteins* (Lang, 2007b) - evaluates IMRs within the gag polyprotein of HIV1 and demonstrates that the five longest IMRs in the polyprotein each translate the key functional feature in each of the (five) mature cleavage products.

The two types of IMRs identified by Lang (2005) are (i) the longest within a sequence (mIMRs) and (ii) those bounded by the nearest downstream reverse dinucleotide (rdIMRs). When the positions of all the repeats within a sequence are determined, it becomes apparent that many spans within a gene consist of IMRs arranged in a hierarchal order - i.e. several IMRs may be nested within a larger IMR. A method to systematically nest IMRs and compare the spans of IMRs to the spans of PSEs was developed during the course of this thesis, and is described in Lang (2005). In these analyses, mIMRs and rdIMRs are independently determined and nested, then compared.

The coincidence of rdIMRs, mIMRs and PSEs were mapped and anlyzed in TnsA, the gag gene of HIV-1 and 16 additional proteins. The co-linearity of IMRs and PSEs was evaluated at various levels of symmetry ranging from ≥ 30% to ≥ 75%. An optimal level of symmetry was found to occur at ≥ 50%; at this level most PSEs were identifiable and the coincidence of PSEs and IMRs was statistically significant. Functional motifs in TnsA were identified using on-line resources and experimental papers. Eighty-eight percent of the known or predicted protein functional motifs in TnsA were found to be the translated product of rdIMRs that are > 16 nt long and ≥ 50% symmetric. The longest mIMRs were observed to occur at sites critical to the function of the protein.

The lengths of IMRs were evaluated systematically in the gag polyprotein of HIV-1 (Lang, 2007b). In this study, all IMRs were identified, then ordered by size. Each of the

5 longest IMRs translates the most significant motif in a different cleavage product (**Table 5**). The length of each rdIMR appears to be related to the functional signficance of the translated segment. The relationship between IMR size and functional significance suggests that selective pressure operates in a punctuated, highly specific manner, on DNA, within a polyprotein, and that selection for IMRs may be a fundamental process of evolution.

At the outset of this work net dinucleotides and mirror repeats were thought to be unrelated entities, a hypothesis proven incorrect by analysis of the distribution of IMRs and PSEs. **Figure 11** illustrates how net dinucleotides and IMRs are related. The sequence (in Figure 11) consists of a long rdIMR that contains two shorter rdIMRs. The span of the largest rdIMR is indicated above the sequence, the grey ***m*** indicating the middle of the repreat and the turquoise ***s*** indicating the terminal dinucleotides. Below the sequence, each of the rdIMRs are illustrated, exhibiting a hierarchal order. Dinucleotides and the reverse dinucleotides define each mirror repeat. The largest mirror repeats for each sequence are determined by evaluating from the 5' to the 3' of the sequence, each rdIMR. The first rdIMR is assigned the designation Level-1. All IMRs that begin and end within it are assigned higher levels (e.g., Level-2, Level-3, etc). A new Level-1 IMR is identified if an IMR terminates beyond the preceding Level-1 IMR. It may begin within the preceding Level-1 IMR, or after it. Level-1 IMRs, identified by this method, coincide with PSEs.

Because IMRs consist, by definition, of dinucleotides and their reverse, they represent the dinucleotides that are removed by the process of determining "net dinucleotides." The sequence elements that remain - those dinucleotides that are not components of mirror repeats - are the net dinucleotides which are combined into circuits. Net dinucleotides, then, measure those dinucleotides that are not directly related to the span of the individual PSEs.

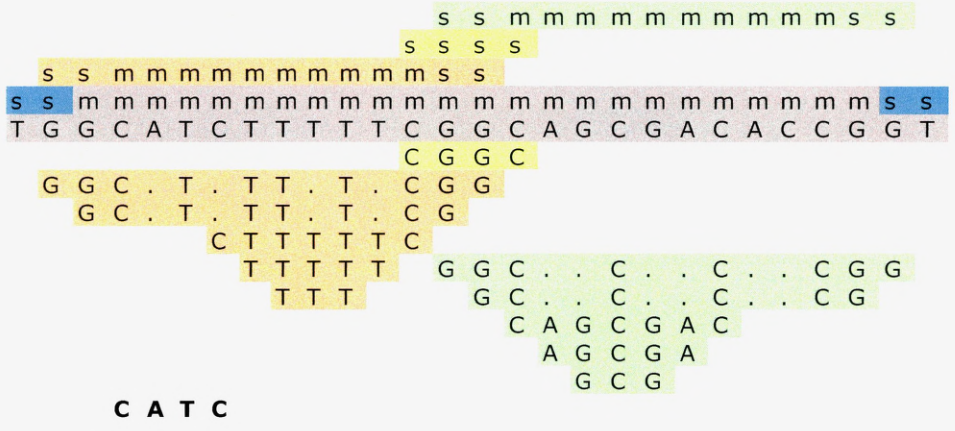


Figure 11. The sequence and span of a large rdIMR (called Level 1) is illustrated in grey, except for the terminal dinucleotides (s) colored turquoise. The large rdIMR contains two smaller rdIMRs (each Level 2 as they are completely contained within a larger rdIMR). For the entire length of this sequence, the only dinucleotides that are not components of a mirror repeat (i.e. do not have an associated reverse dinucleotide) are C A T C (indicated on the lower left of the figure), giving the entire sequence the net value of CATC. The criteria for symmetry is ≥50% symmetric.

The identification of the coincidence of IMRs and key functional motifs in TnsA (Lang, 2005) suggested that IMRs promote the conservation of key functional motifs. This hypothesis was evaluated in a more complex context in the gag polyprotein of HIV-1 (Lang, 2007b). This paper demonstrates that within the gene that translates a polyprotein, selection occurs for both the entire gene and within each segment that translates a mature gene product - i.e. simultaneously at multiple levels of genome organization (**Table 5**). The magnitude and specificity of this selection indicates that it is likely to be a global feature of DNA selection and therefore, a constraining factor in molecular evolution.

The specificity with which IMRs and functional motifs coincide is illustrated by the relationship of IMRs to the cys-his boxes in the (gag) nucleocapsid, illustrated in **Figures 12 and 13.**

Rank	mIMR	prot	len	DNA positions	AA positions	Structure or function
1	#1-gag	MA	95	0270-aa..ca-0364	091-RI..DT-122	MA-H5 related to viral entry
2	#2-gag	CA	87	0742-gg..tg-0828	248-GW..RM-276	CA-H7 longest constituent of viral core
3	#3-gag	p1	85	1256-aa..aa-1340	419-EG..GN-447	end NC, p1 to p1-p6 cleavage site
4	#6-gag	NC	81	1171-aa..ga-1251	391-KC..CG-417	1st cys-his box; EF1α binding
5	#7-gag	p2	80	1065-ac..ca-1144	356-PG..GN-382	p2, critical to budding
6	#10-gag	CA	76	0812-at..ta-0887	271-NK..DY-296	major homology region
7	#11-gag	CA	75	0920-ag..ga-0994	307-EQ..KT-332	endocytosis signal 1; CA-H9 helix
8	#14-gag	CA	69	0985-ga..ag-1053	329-DC..QG-351	endocytosis signal 2; CA-H10 helix
9	#15-gag	CA	64	0543-ca..aa-0606	181-PQ..LK-202	CA-H3-H4 helices, part of viral core
10	#17-gag	NC	64	1362-gc..ag-1425	455-PT..QK-475	L-domain (budding); Tsg101
					455-PT..QK-475	docking; ubiquitin-gag conjugate
11	#1-NC	NC	59	1209-aa..aa-1267	404-NC..QM-423	2cd cys-his box; end NC
12	#2-NC	NC	47	1153-aa..ca-1199	385-NQ..GH-400	EF1α binding

Table 5. This table lists IMRs by rank in the gag protein of HIV-1. The heading "prot" indicates the mature protein cleaved from the gag polyprotein (MA=matrix, CA=capsid, p1=spacer 1, NC=nucleocapsid, p2=spacer2). The five longest mIMRs translate the longest IMR in each of the mature protein products, which are also the most significant motifs in each protein.

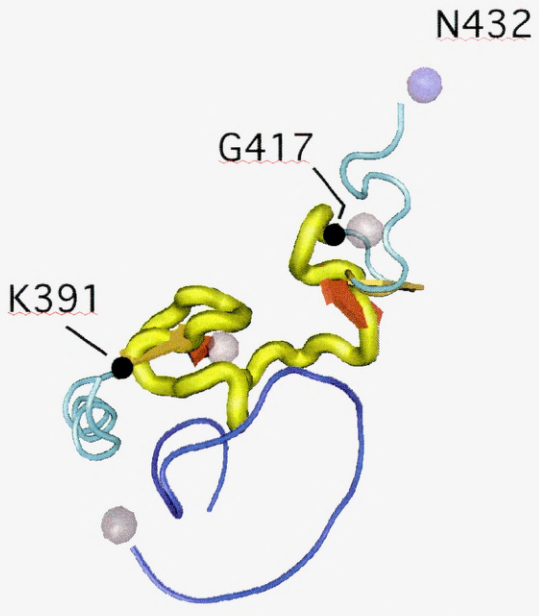


Figure 12. The largest mIMR in the nucleocapsid. - #6-gag spans both zinc knuckles and the spacer between them. Each of the next largest mIMRs in the NC, translates one of the Cys-His boxes.

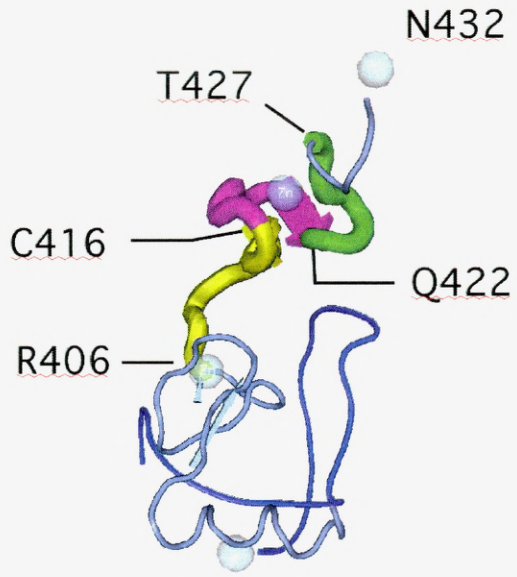


Figure 13. This figure is in the same polar orientation as A and B, but rotated. It illustrates the two longest rdIMRs in Gag that occur in the nucleocapsid, which overlap. Within the overlap region (in purple) two amino acids bind the zinc ion.

5. The composition of the third codon position varies between different types of protein structural elements in the translated protein product.

One of the initial objectives of this thesis was to explore the concept of codon degeneration. Despite the technical explanation of the wobble hypothesis (Crick, 1966), the idea of codon degeneration seemed disturbing due to the significance of amino acid order on protein function and identity. The dinucleotide equivalent of the ordered connection between amino acids is the third codon position of one amino acid and first codon position of the next. Given the high degree of specificity exhibited by most genes and proteins, the idea that this dinucleotide pair included an insignificant component seemed out of character.

The experimental design for this project is based on the detailed analysis of 17 proteins for which the tertiary structure has been experimentally determined. An EXCEL spread sheet was used for each protein, to combine the DNA sequence, protein sequence, and protein structure determination. The protein structural elements identified were: β–turns, β–sheets, α-helices, g-helices and unspecified. Equations were constructed within the EXCEL spreadsheets to tally the nucleotide composition at each codon position, and

the type of protein structural element. Secondary structure positions by PROMOTIF (Hutchinson and Thorton, 1996) and Kabsch and Sander (1983) were also included on the spreadsheet. The tallied results were evaluated for statistical reliability using Fisher's exact test (Langsrud, 2007).

The results of this analysis, for statistically significant associations, are summarized in **Table 6**. Certain nucleotides occur more frequently for particular combinations of protein structural element and codon position. In unstructured segments ('n' in Table 6), A or C are most common in the 1st codon position, and A or G in the second codon position; there are no preferential nucleotide types in the third codon position. The only distinguishing codon preference of S-bends is the use of A in the second codon position. Beta-sheets have characteristically high T in the second codon position, and C or T in the third codon position. In contrast, β–turns have A or G in the second codon position and A or C in the 3rd codon position. Helices are characterized by A or G in the 3rd codon position. Although the association between type of PSE and type of nucleotide does not reach statistically significant values for every protein, the trends are distinctive, and support the hypothesis, that there is an association between the nucleotide composition of each codon position and the type of protein structural element.

At about the time this work was being summarized, a paper that described the relationship between nucleotide composition of the third codon position and PSEs was published (D'Onofrio *et al*, 2002). These studies are in good agreement with the work discussed in this thesis.

Summary 14 genes, 2334 nt	PSE	position 1 A	C	A or C	G	T	position 2 A or G	A	C	G	T	position 3 A or G	A	C	A or C	G	T	C or T
helix	H											x						
turn	T						x								x			
b-sheet	E					x					x							x
S-bend	S							x										
none	n			x			x											

For each gene	PSE	position 1 A	C	A or C	G	T	position 2 A or G	A	C	G	T	position 3 A or G	A	C	A or C	G	T	C or T
alpha-catenin	H										0.024							
Aif	H																	
procapsase-7	H											**0.068**				0.068		
ARSB	H	0.047										**0.028**	0.028					
Brca1	H											**0.012**	0.012					
CNN1	H									0.039	0.069							
CDC42	H				0.031							**0.083**				0.083		
COL10A1	H																	
Cdk2	H					0.015					0.047							
HIF	H							0.018				**0.038**				0.038		
IL19	H																	
lysozyme C	H											**0.038**				0.038		
thioredoxin	H																	
TnsA	H				0.076							**0.028**	0.028					
Vcam-1	H																	
alpha-catenin	T					0.088												
Aif	T	0.042					**0.017**	0.017										
procapsase-7	T				0.095		**0.073**			0.073				0.006	**0.006**			
ARSB	T																0.062	
Brca1	T												0.612		**0.612**			
CNN1	T																	
CDC42	T												0.067		**0.067**			
COL10A1	T						**0.022**			0.022								
Cdk2	T						**0.032**		0.043	0.032								
HIF	T													0.010	**0.010**			
IL19	T										0.062							
lysozyme C	T		0.056				**0.040**	0.040						0.039	**0.039**			
thioredoxin	T						**0.008**			0.008								
TnsA	T																	
Vcam-1	T				0.043		**0.012**	0.012										
alpha-catenin	E																	
Aif	E																0.000	**0.000**
procapsase-7	E					0.005					**0.000**							
ARSB	E										**0.000**							
Brca1	E										**0.001**						0.035	**0.035**
CNN1	E																	
CDC42	E																0.089	**0.089**
COL10A1	E					0.045					**0.028**							
Cdk2	E										**0.013**							
HIF	E										**0.009**							
IL19	E																	
lysozyme C	E		0.063											0.072				**0.072**
thioredoxin	E										**0.007**							
TnsA	E													0.076				**0.076**
Vcam-1	E					0.064												
alpha-catenin	S																	
Aif	S							**0.022**		0.009								
procapsase-7	S				0.071			**0.010**									0.080	
ARSB	S							**0.025**										
Brca1	S																	
CNN1	S																	
CDC42	S																	
COL10A1	S							**0.089**			0.063							
Cdk2	S																	
HIF	S																	
IL19	S																	
lysozyme C	S																	
thioredoxin	S																	
TnsA	S																	
Vcam-1	S																	
alpha-catenin	n	0.076		**0.076**			**0.018**		0.088	0.018								
Aif	n																	
procapsase-7	n		0.028	**0.028**														
ARSB	n																	
Brca1	n	0.020		**0.020**							0.002			0.003				
CNN1	n						**0.014**			0.014								
CDC42	n																	
COL10A1	n		0.031	**0.031**														
Cdk2	n																	
HIF	n				0.026		**0.004**			0.004						0.040		
IL19	n						**0.043**	0.043								0.026		
lysozyme C	n																	
thioredoxin	n	0.046		**0.046**										0.071				
TnsA	n						**0.092**	0.092										
Vcam-1	n																	

Table 6. Certain types of nucleotides are preferred for each type of PSE at each codon position. The p-values for these associations were determined by Fisher's exact test (Langsrud, 2007) and are indicated for each position for each of fifteen proteins. The accession number for each of these genes is provided in Lang (2005).

6. Net dinucleotide composition characterizes gene function at multiple levels of genome organization

Net dinucleotides were first evaluated as a wild guess more than twelve years ago - on tRNAs because they were short, and the tRNAs of *E. coli* because one tRNA of each species had been determined. The counts were made using an EXCEL spreadsheet. This sample set exhibited the same properties which have since been confirmed for multiple organisms (unpublished analyses), including comprehensive sets of tRNAs from whole genomes. The circuit assemblage (CA) of each species of tRNA is unique, i.e. distinguishes one tRNA species from another within the organism. This highly specific property suggested that CAs might also be related to function in other DNA sequences, including protein-coding DNA.

A database that was intended to investigate this property (described in section 3e of this paper) provided further support of this hypothesis but also indicated that the property was species-specific. The validity of the hypothesis is demonstrated within a single species by the analysis of the CAs of the proteins of HIV-1 (Lang, 2007a). Despite the variability of the genome (up to 30% sequence variability within 110 genomes from different subtypes), the CA of each HIV gene distinguishes it from the others.

CAs also distinguish between functional components of genes - as demonstrated by the differences between the hypervariable loops and conserved segments of the env gene. In **Table 7**, the functional components of *env* are compared with groups of human hormones and their receptors. This table demonstrates that certain CAs are characteristic of hormones and receptors, which differ from each other by the quantity of different circuits. Some of the functional segments of env - V1V2 and C3 - have the same type of circuits as insulin. Others have the same type of circuits as CD4, with which the env gene is known to interact (Schmittman *et al*, 1998). It seems likely that CAs can be used to group gene families within an organisms and provide familial information for hypothetical proteins that is currently unavailable by any other means.

The potential for CAs to discriminate between organisms was evaluated by comparing whole genomes of viruses (**Table 8**). This data suggests that CAs also have the property of functional discrimination at the genomic level.

Line	Accession Number	Hormone / Receptor	GCATG	GCTAG	GCAG	GCTG	GATG	SUM	RATIO	expressed by
1	Sample Set Average	HIV-1, complete genome	34		186	93		313		
2	M35160	CD4	15		15	31		61		
3	K03455	env - gp120	11		28	42		81		
4		env - signal sequence	1			1	4	6		sim to somatostatin
5		env - c1	5		2	7		14		sim to CD4
6		env - V1V2		3	1	4		8		sim to insulin
7		env - C2	4		9			13		
8		env - V3	1		2			3		
9		env - C3		1	1	1		3		sim to insulin
10		env - V4	1		1	1		3		sim to CD4
11		env - C4-V5-C5	3		4	3		10		sim to CD4
12		env - C1 + V4 + C4-V5-C5	9		7	11		27	2.3	all segs sim to CD4
13	M35160	CD4	15		15	31		61		
		Human Hormone-Receptor Pairs								
14	M26095	calcitonin	9		14	9		32	2.1	thyroid
15	BC075028	calcitonin receptor	34			14	20	68		
16	NM_001048	somatostatin	5			17	1	23	2.5	stomach-intestine
17	L14856	somatostatin receptor	12		9	36		57		
18	BC101843	oxytocin	1		1	22		24	2.7	posterior pituitary
19	NM_000916	oxytocin receptor	22		16	27		65		
20	J00152 + AL109660	CGA (thyroid-stimulating-alpha)	11		4	12		27	3.1	anterior pituitary
21	NM_00369	TSH receptor	40			39	5	84		
22	J00152 + NM_000510	CGA + FSHB (follicle-stimulating B)	5		11	12		28	3.2	anterior pituitary
23	NM_181446	FSHR receptor	75		4	11		90		
24	NM_000948	prolactin	18			3	3	24	3.4	anterior pituitary
25	NM_000949	prolactin receptor	38			36	8	82		
26	J00152 + NM_000894	CGA + LHB (luteinizing hormone)	10		7	9		26	3.7	anterior pituitary
27	S57793	LHR-receptor	45			40	11	96		
28	J00152 + NM_033377	CGA + gonadotropin8	10		5	7		22	3.9	hypothalamus
29	NM_000233	LHCGR-receptor	42			31	12	85		
30	J00265	insulin		2	2	24		28	4.5	pancreas
31	M10051	insulin receptor	35			21	70	126		
		Other Signalling Proteins								
32	L11284	MEK1	37		6	10				
33	L11285	MEK2	26			1	5			
34	X60188	ERK1	16			20	5			
35	AF135158	ERK2	64		15	29				

Table 7. Two of the circuits for hormones and their receptors are similar, but most hormone-receptor pairs also have a unique and complementary circuit (GCAG-GATG). The env sequence of gag contains hypervariable loops (V1-V5) and conserved regions (C1-C5). Three of these functional domains (V1V2 and C3) have a CA similar in composition to that of insulin.

Group*	organism	length	ACTA	GCATG	GCTG	GATG	GCAG	GACTG	ATCA	GCTAG	GTAG	GTACG	GACG	GATCG	tag
viroids															
NC_003463	Apple Dimple Fruit Viroid	306	**3**										2		GTAC
NC_001340	Apple Scar Skin Viroid	329	1								1	**4**			GTAC
NC_003777	Apple Fruit Crinkle Viroid	371	1							**4**	2				TAC
NC_001651	Citrus Bent Leaf Viroid	315	**4**		1					1					CT
NC_001907	Citrus viroid Ia	326	**5**										1		CT
+ssRNA															
NC_001612	Human enterovirus A	6582		**128**	37		15								AG
NC_001489	HAVgp1, Hepatitis A	6684		**171**		74			14						no
-ssRNA															
NC_005222	Hantaan virus segment L	6456		**113**	6		62								AG
NC_005219	Hantaan virus segment M	3408		**63**	16		9								AG
NC_005218	Hantaan virus segment S	1290		**36**	6		5								no
dsRNA															
NC_002063	Leishmania RNA virus 1-1	5222		55	**61**	17									ATG
NC_006431	Cryphonectria_hypovirus4	8457		16	60	**97**									ATG
NC_006276	WCCV-1_clover	1464			**30**	22		3							no
NC_003745	yeastVirus_L-A	4517	2		**40**			14							no
NC_003607	HelminthosporiumVictoriaeVirus	4628				23		**32**					3		no
NC_001492	Cryphonectria hypovirus	11366				**128**			110					20	no
NC_00764	OryzaRufipogonEndornavirus	13884		127		**187**			39						no
NC_003555	GLVgp1, Giardia lamblia virus	5612	**12**									3	8		ACG
ssDNA															
NC_007455	Human bocavirus	4819		46	**75**	20									no
NC_004306	Vibrio cholerae 0139 fs1 phage	6180		27	**44**	40									no
AF495467	Canine minute virus	4616		13	1	**67**									no
M14363	Bovine parvovirus	4567		24	35	**48**									no
NC_001331	Pseudomonas phage Pf1	7147		3	3	**137**									no
dsDNA, no RNA stage															
NC_001460	Human adenovirus A	32461		74	**397**		206								GCT
NC_001576	Human papilloma virus type 10	7233		39	**116**		33								no
RT															
110 sequences	HIV-1 - sample set	8626±39		34±7	93±8		**186±9**								no
NC_001426	HumanT-lymphotropicVirus1	5846		1	89		**103**								no
NC_001407	Rous_sarcoma_virus	8330		44	**113**		26								no
NC_001654	RT_bovine_HIV	7313		50	**101**		28								no
NC_001482	RT_Feline_immunodeficiency_virus	8543		**90**	61		57								AG
NC_001408	RT_Avian_leukosis_virus	4811		**45**	**46**		7								no
NC_001725	Strawberry_vein_banding	7022		**56**	30		37								AG

* ds = double-stranded; ss = single stranded; + is positive-strand; - is negative-strand; RT = retro-transcribing

Table 8. CAs of viruses characterize their mode of replication. Only a single strand of the dsRNA viruses were analyzed, likely leading to the less consistent circuit values for these organisms.

7. Conclusions and implications

The analyses that comprise this thesis reveal new perspectives of genic and genomic organization that have practical applications and also suggest the existence of at least one previously unrecognized mechanism that strongly impacts molecular evolution. The findings in these thesis studies indicate that the sequences that comprise genes and genomes are highly ordered structures and that this ordering conserves the species identity of the organism and the functional identity of its protein coding genes. The ordering is pervasive and ancestry of all sequences is traceable through both its parts and its whole, indicating that the mechanism that promotes sequence organization conserves both the character of the species and the character of the function.

Generations of researchers have demonstrated that dinucleotide composition is conserved, and that it is a solid criteria for differentiating species. The selection of the "most fit" sequence modifications arising from any of the multiple mechanism of molecular evolution would result in considerable nucleotide and dinucleotide drift over evolutionary time if some mechanism did not exist to counteract drift. The ability to construct phylogenies of both genes and genomes demonstrates that mechanisms or constraints exist that conserve nucleotide and dinucleotide identity. These mechanisms or constraints are beyond those of DNA repair as they operate within organisms that exhibit poor DNA repair systems, such as HIV (Sharp *et al*, 1999).

Although at this time a mechanism for the conservation of dinucleotide composition has not been postulated, it would function to compensate for mutations in a sequence by the addition or modification of nucleotides which would result in the preservation of species identity (dinucleotide composition) and gene function. The consistency and distinction of net nucleotide circuits suggest that they are evidence of an ongoing mechanism that leads to preservation of net dinucleotide values and dinucleotide identity and function. This mechanisms would function by promoting compensatory changes in a sequence. The affect of this mechanism would be similar to what is observed in most sequences - a significant number of dinucleotide-reverse-dinucleotide pairs - and a variable number of unpaired dinucleotides (which constitute circuits). The analyses in this thesis demonstrate that circuit assemblages within a species are conserved for particular

functions. Therefore this mechanism, though as yet undetermined, must be fundamental to the process of evolution.

The hierarchal nature of imperfect mirror repeats and the relationship of them to protein structural elements and key functional sites in proteins suggests that functional success and functional consistency in the protein is associated with ordered dinucleotide pairs in the genes and genome. Thus, imperfect mirror repeats seem likely to contribute toward fitness and likely become selected by fitness, but primarily due to their relationship to structural components. Those dinucleotides that are not components of IMRs, and thus not associated with protein structural elements, comprise net dinucleotide circuits which characterize both species and function.

A DNA walk is a graphic representation of a DNA sequence. DNA walks are distinctive for each organism, differentiate genes within an organism, and distinguish introns from exons. The trajectory of the DNA walk becomes convoluted and noisy for protein-coding segments of DNA which consist of short range DNA circuits. These clusters inherently form as a consequence of dinucleotide and reverse-dinucleotide pairs, and net dinucleotide circuits within protein-coding segments. Future work that combines the quantitative values of the DNA walk with qualitative values of circuits may lead to new perspectives of genome organization.

Although the basis of the relationship between net dinucleotide circuits and species identity and gene function has not been identified, circuits can be used to find relationships between genes and within genomes that have not been identified by existing methods. Each species of tRNA within an organism is unique by these values, and each of their synthetases. The possibility exists that circuit assemblages can qualitatively quantify the biochemical networks within each species.

The work contained in this thesis demonstrates that CAs (derived from dinucleotide composition) discriminate between genomes, thereby providing a species-specific identity for each genome. Within genomes, CAs identify proteins or segments of proteins that are functionally related or not. These methods, then, provide an intrinsic identification of protein function and structure rather than a comparative analysis, and, additionally, new perspectives of evolutionary processes including both genomic organization and speciation.

This work differs fundamentally from current methods of predicting the secondary structure of proteins and protein functional motifs, and provides new perspectives and advantages. Current methods that predict protein function rely solely on amino acid composition and position, and sequence comparison to other proteins. The spans of PSEs are predicted based on the amino acid composition of adjacent amino acids over localized regions. Protein motifs are identified based on the punctuated distribution of specific amino acids. The prediction programs for these features (described previously in this paper) often give slightly different results. By current methods, it is difficult to predict from secondary structure, the impact of DNA and protein mutation on protein function, except for cases where DNA mutation causes an amino acid mutation that has a highly specific impact, e.g. a change in charge, the relationship of DNA to protein structure and function has been unexplored. The work related to IMRs described in this thesis describes a precise and consistent method to predict the span of PSEs and protein functional domains, and a method of ranking their significance to the function of the protein. The authentication of these methods provides a new basis from which to develop new types of predictive models. This work also demonstrates that evolutionary processes select for DNA IMRs and thereby constrain the amino acid composition of a protein, and ultimately its function.

The concepts of IMRs and net dinucleotides challenge some long-accepted tenets of the process of evolution. The most widely accepted theories regard mutation and evolution as random processes, impacted by on-going selection for fitness within environmental niches that ultimately lead to speciation. Essential to the underlying premise of randomness is that there is no discernible interconnectedness within a gene, e.g. one cannot predict downstream sequence and composition. This would not be the case if gene function were related to conserved net dinucleotide values. Nor, if random processes prevail, should there exist a "net dinucleotide value" that remains consistent for a species despite extreme diversity - as for example in quasispecies.

The work contained in this thesis demonstrates that functional units of genes and genomes are highly ordered structures that maintain characteristic values of net dinucleotide composition despite mutation and evolution. The mechanisms that maintain

this structure counteract the pressures of random evolutionary events to maintain function- and species-specific dinucleotide and net dinucleotide composition.

The thesis studies also demonstrate that some of the key attributes of proteins can be determined from the DNA that transcribes them. These attributes include the span of individual PSEs, the span of key protein functional domains and the location of key functional motifs in the protein. Net dinucleotide composition characterizes and distinguishes functional segments of DNA at all levels of genomic organization. Together, the work described by these studies strongly suggests that selection for IMRs and conservation of net dinucleotide composition are previously unrecognized constraining forces in molecular evolution.

The methods described in these studies provide new insights into evolution, speciation and gene function, and several tools to improve the prediction of significant functional sites in proteins. The comparative aspects of genome and protein identity provided by CAs do not require sequence alignment. They rely on intrinsic properties of the sequence, composition combined with order (dinucleotides) to achieve a specificity that can be summarized concisely. CAs have been shown to be consistent with traditional phylogenetic methods (tree-building), and therefore may be useful for the identification of familial groups of proteins, the functional classification of unknown and hypothetical proteins, and analysis of biochemical pathways.

Analysis of IMRs by the methods described make it possible to identify key protein domains and individual protein structural elements from a DNA sequence. This analysis also provides a method of ranking motifs according to their functional relevance and are therefore useful for the determination of key motifs in tertiary structures.

The use of IMRs for the prediction of the affects of DNA mutation on protein structure depends on future work that determines whether the association of IMRs with protein structural elements is an on-going, immediate affect, as for example, involved with the rate of passage of mRNA through the ribosome and subsequent folding of the nascent protein, or whether it is an affect of selection processes operating over evolutionary time.

The studies comprising this thesis address concepts of molecular biology that have seemed, to this writer, to be inconsistent or unfounded. (1) The idea that codons are "degenerate," i.e. the composition of the third codon position does not specify some

specific DNA or protein attribute seems inconsistent with the precision and interdependence that exists in all aspects of molecular biology. (2) The assumption that there is no information about protein structure within DNA sequences other than that which can be inferred from the translated protein. (3) The idea that responses of an organism to an environmental challenge are solely attributed to Darwinian selection. If new or modified proteins can be produced by an organism in response to external stimuli, cellular mechanisms must exist that can initiate adaptive change within the genome. New studies of rapid adaptive change (< 15 generations) have concluded that the basis of genomic change is directed rather than random mutation (Hastings *et al*, 2000; Rosenbery *et al*, 2004).

The new types of sequence analysis that I have been developing can be used to test most of the above-mentioned topics. These new methods are based on the use of every nucleotide and arrive at a new type of quantitative measure that intrinsically combines both DNA and protein structure and function.

8. Future Work

Many of the shapes and structures found in organisms have counterparts in mineral structures and can be described mathematically in relatively simple terms. Minor changes in rates of biologic processes can produce significantly different shapes and structures in organisms, and are evidence that simplistic principles often underlie what appear to be complex differences. The work described in this thesis was undertaken with these concepts at the forefront. My objective in studying genes was to evaluate whether the combination of nucleotide position and composition could become mathematic variables that could be integrated into biochemical calculations. In my opinion, this is still an achievable objective, and the works contained in this thesis are essential and requisite steps towards achieving that goal. This work establishes the relevance of DNA composition and position to protein function. Because of the novelty of this paradigm, the supporting research focuses on features that are indisputable, on a scale that can be readily tested.

The new methods described and documented by this thesis and its associated publications have broad applications, and preliminary investigations of many of them have been made. CAs have been tested for genes that are constitutents of pathogenic secretion

loci in bacteria. Within this operon, CAs can ascribe function to hypothetical proteins and identify different components of secretion systems. CAs were found to differentiate developmental processes in the sea urchin *Strongylocentrotus purpuratus* (http://sugp.caltech.edu/endomes/index.html). Functional categories in the genes of these pathways corresponded with many of those in the CA database that was developed for this thesis. CAs also seemed to provide a basis for predicting gene interaction within these pathways. It is anticipated that CAs will have broad applications for the identification of hypothetical genes and pathway analysis, provided that organism-specific databases are constructed.

The relationship between IMRs and protein functional motifs has been sufficiently established by the work described in this thesis, to begin to develop and test models that predict the affect of DNA mutation on protein function. A preliminary study towards this goal was made using the Sarcomere Protein Gene Mutation Database (http://genetics.med.harvard.edu/~seidman/cg3/index.html), which provides information about the affect (and its severity) of each DNA mutation in human myosin. IMRs were evaluated for several of these mutations, and in most cases, the loss of a long IMR was associated with a mutation having a severe affect on protein function. It is anticipated that comparison of these factors in several databases will provide enough data to develop a statistically reliable model.

9. Acknowledgements

This work was made possible by the dedication and patience of Professor John Palfreyman from whom I learned to think critically and write clearly. I also appreciate the support and encouragement of the many members of the staff and faculty of the University of Abertay-Dundee, particularly Louis Natanson, Carol Conway, Jonathan Teppett, Lynn Christie, Dawn Keen and Carol Panton. This work was also made possible by my brother Doug MacLean who provided me a place to stay, a computer and technical support, between jobs, and while I completed my thesis. Thank you all very much.

10. Addendum

The existing study describes two novel methods of sequence analysis that provide new perspectives of gene and genome organization and identity. The examples in this thesis demonstrate that the results of these novel methods are consistent with the results achieved by traditional analyses. The extension of these novel methods into a predictive capability would require the development of confidence scores for the relevant parameters of each method.

The first method produces a hierarchal ordering of DNA imperfect mirror repeats (IMRs) within which the longest mirror repeats were found to occur at key functional sites in the translated protein. Current methods of identifying key functional sites rely on comparisons with databases of known functional motifs. The new methods described by this thesis have the additional potential to identify new motifs and other types of segments most essential to the function of the protein for DNA of unknown origin and proteins that have undetermined structure(s). A key to validating these predictions is to develop confidence scores for the distribution of mirror repeats. To achieve this frequency distributions will be calculated for the total number of repeats, the length of all repeats and the percent symmetry of all repeats based on a comparison of well-documented protein-coding genes and random sequences of the same composition. These analyses will also include summaries of the relationship between key functional motifs and the length of IMRs. Reliably and richly annotated sequences will be used for this study.

The second method identifies a small subset of dinucleotides - net dinucleotides and circuits – which are equal to approximately one percent of the total sequence length. Net dinucleotides were found to exhibit segregation (i.e. identity) properties that correspond with traditionally determined phylogenetic and functional properties of genes and genomes. In order to expand the application of this method to a predictive methodology, a means will be determined to disqualify components of a set. For example, all endonucleases analyzed to date consist of 20-30 circuits of GCTG-GATG-GCATG; what is/are the cut-off(s) that indicate(s) the gene is not an endonuclease? A preliminary determination of these cut-offs will be determined by whole-genome analysis of the net dinucleotide composition of several bacterial genomes, including several strains from the

same species and several from related species. The results of this analysis will be compared with those of existing methods such as COG classification systems (gene function classification).

11. References

Abramson G, Alemany PA, Cerdeira. 1997. Noisy Levy walk analog of two-dimensional DNA walks for chromosomes of S. cerevisiae. Phys Rev E. 58:914-918.

Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ. 1990. Basic local alignment search tool. J Mol Biol. 215(3):403-10.

Andrieu O, Fiston AS, Anxolabéhère D, Quesneville H. 2004. Detection of transposable elements by their compositional bias. BMC Bioinformatics. Jul 13;5:94.

Arndt PF, Hwa T. 2005. Identification and measurement of neighbor-dependent nucleotide substitution processes. Bioinformatics. 21(10):2322-8.

Beerman TA, Lebowitz J. 1973. Further analysis of the altered secondary structure of superhelical DNA. Sensitivity to methylmercuric hydroxide, a chemical probe for unpaired bases. J Mol Biol 79, 451-470.

Berezikov E, Guryev V, Plasterk RHA, Cuppen E. 2004. CONREAL: Conserved regulatory elements anchored alignment algorithm for identification of transcription factor binding sites by phylogenetic footprinting. Genome Res 2004, 14:170-178.

Bernardi A, Gaillard C, Bernardi G. 1975. The specificity of five DNAases as studied by the analysis of 5'-terminal doublets. Eur J Biochem. 52(3):451-7.

Blanchard KL, Fandrey J, Goldberg MA, Bunn HF. 1993. Regulation of the erythropoietin gene. Stem Cells. 1993 May;11 Suppl 1:1-7.

Blanchette M, Tompa M. 2002. Discovery of regulatory elements by a computational method for phylogenetic footprinting. Genome Res 2002, 12:739-748.

Bland C, Ramsey TL, Sabree F, Lowe M, Brown K, Kyrpides NC, Hugenholtz P. 2007. CRISPR recognition tool (CRT): a tool for automatic detection of clustered regularly interspaced palindromic repeats. BMC Bioinformatics. 2007 Jun 18;8:209.

Borstnik B and Pumpernik D. 2002. Tandem repeats in protein coding regions of primate genes. Genome Research 12:909-915.

Bray N, Pachter L. 2003. MAVID multiple alignment server. Nucleic Acids Res. 31(13):3525-6.

Cebrat S and Dudek MR. 1998. The effect of DNA phase structure on DNA walks. European Physical Journal B. 3:271-276.

Cox R, Mirkin SM. 1997. Characteristic enrichment of DNA repeats in different genomes. Proc Natl Acad Sci USA 94:5237-5242.

Crick FH. 1966. Codon--anticodon pairing: the wobble hypothesis. J Mol Biol. 19(2):548-55.

Das MK and Dai HK. 2007. A survey of DNA motif finding algorithms. BMC Bioinformatics. S21. doi:10.1186/1471-2105-8-S7-S21.

Deb S, DeLucia AL, Baur CP, Koff A, Tegtmeyer P. 1986. Domain structure of the Simian Virus 40 core origin of replication. Mol Cell Biol 6(5):1663-1670.

Desvergne B. 1994. How do thyroid hormone receptors bind to structurally diverse response elements? Mol Cell Endocrinol. 100(1-2):125-31.

Domaniç NO, Preparata FP. 2007. A novel approach to the detection of genomic approximate tandem repeats in the Levenshtein metric. J Comput Biol. 14(7):873-91.

D'Onofrio G, Ghosh TC, Bernardi G. 2002. The base composition of the genes is correlated with the secondary structures of the encoded proteins.Gene. 2002 Oct 30;300(1-2):179-87.

Edelmann L, Spiteri E, Koren K, Pulijall V, Bialer MG, Shanske A, Goldberg R, Morrow BE. 2001. AT-rich palindromes mediate the constitutional t(11;22) translocation. Am J Hum Genet. 68(1):1-13.

Eickbush TH. 1992. Transposing without ends: the non-LTR retrotransposable elements. New Biol 4:430-440.

Forsdyke DR. 1995. Relative roles of primary sequence and (G + C)% in determining the hierarchy of frequencies of complementary trinucleotide pairs in DNAs of different species. J Mol Evol. 41(5):573-81.

Gentles AJ, Karlin S. 2001. Genome-scale compositional comparisons in eukaryotes. Genome Res. 2001 Apr;11(4):540-6.

Gierer A. 1966. Model for DNA and protein interactions and the function of the operator. Nature (London) 212, 1480-1481.

Green H and Wang N. 1994. Codon reiteration and the evolution of proteins. Proc Natl Acad Sci USA 91:4298-4302.

Grissa I, Vergnaud G, Pourcel C. 2007. CRISPRFinder: a web tool to identify clustered regularly interspaced short palindromic repeats. Nucleic Acids Res 35(Web Server issue):W52-7.

Guo AM. 2007. Long-range correlation and charge transfer efficiency in substitutional sequences of DNA molecules. Phys Rev E Stat Nonlin Soft Matter Phys. June 75(6 Pt 1):061915.

Gupta R, Sarthi D, Mittal A, Singh K. 2007. A novel signal processing measure to identify exact and inexact tandem repeat patterns in DNA sequences. EURASIP J Bioinform Syst Biol. 43596.

Hammock EAD and Young LJ. 2005. Microsatellite instability generates diversity in brain and sociobehavioral traits. Science 308:1630-1634.

Heilman-Miller SL,Wu T, Levin JG. 2004. Alteration of nucleic acid structure and stability modulates the efficiency of minus-strand transfer mediated by the HIV-1 nucleocapsid protein. J. Biol. Chem. 279(42):44154-65.

Henney H, Storck R. 1963. Nucleotide composition of ribonucleic acid from *Neurospora crassa*. J Bacteriol. 85:822-6.

Heringa J. 1998. Detection of internal repeats: how common are they? Curr. Opin Struct Biol. 8(3):338-45.

Hofacker IL. 2003. Vienna RNA secondary structure server. Nuc Acids Research. 31(13):3429-3431.

Hutchinson EG and Thornton JM. 1996. PROMOTIF--a program to identify and analyze structural motifs in proteins. Protein Sci. 5(2):212-20.

Jiang B, Zhang MQ, Zhang X. 2007. OSCAR: one-class SVM for accurate recognition of cis-elements. Bioinformatics. 23(21):2823-8. Epub 2007 Oct 5.

Kabsch W and Sander C. 1983. Dictionary of protein secondary structure: pattern recognition of hydrogen-bonded and geometrical features. Biopolymers 22(12):2577-637.

Kankainen M, Löytynoja A. 2007. MATLIGN: a motif clustering, comparison and matching tool. BMC Bioinformatics. 8:189.

Karlin S. 1998. Global dinucleotide signatures and analysis of genomic heterogeneity. Curr Opin Microbiol. 1(5):598-610.

Karlin S, Ladunga I. 1994. Comparisons of eukaryotic genomic sequences. Proc. Natl. Acad. Sci U S A. 91(26):12832-6.

Karlin S, Campbell AM, Mrázek J. 1998. Comparative DNA analysis across diverse genomes. Annu Rev Genet. 1998;32:185-225.

Karlin S, Mocarski ES, Schachtel GA. 1994. Molecular evolution of herpesviruses: genomic and protein sequence comparisons. J Virol. 68(3):1886-902.

Kerr DJ and Workman P. 1994. *New Molecular Targests for Cancer Chemotherapy.* CRC Press, Boca Raton, FL, USA. 236 pp.

Kieff E, Dambaugh T, Heller M, King W, Cheung A, van Santen V, Hummel M, Beisel C, Fennewald S, Hennessy K, Heineman T. 1982. The biology and chemistry of Epstein-Barr virus. J Infect Dis. 1982 Oct;146(4):506-17.

Kumar S, Tamura K, Nei M. 1994. MEGA: Molecular Evolutionary Genetics Analysis software for microcomputers. Comput Appl Biosci. 10(2):189-91.

Kurtz S, Choudhuri JV, Ohlebusch E, Schleiermacher C, Stoye J, Giegerich R. 2001. REPuter: The Manifold Applications of Repeat Analysis on a Genomic Scale. Nucleic Acids Res., 29(22):4633-4642.

Lander ES, *et al.* 2001. Initial Sequencing and Analysis of the Human Genome. Nature. 409: 860-921.

Lang DM. 2000. Net nearest neighbor analysis (NNNA) summarizes non-compensated dinucleotides within gene sequences. Bioinformatics. Mar;16(3):212-21.

Lang DM. 2005. Imperfect DNA mirror repeats in *E. coli* TnsA and other protein-coding DNA. Biosystems. 81(3):183-207.

Lang, DM. 2007a. Circuit assemblages derived from net dinucleotide values provide a succinct identity for the HIV-1 genome and each of its genes. Virus Genes. Nov 7. PMID:17987374.

Lang, DM. 2007b. Imperfect DNA mirror repeats in the gag gene of HIV-1 (HXB2) identify key functional domains and coincide with protein structural elements in each of the cleaved mature proteins. Virology J. Oct 26:4(1):113. PMID:17963512.

Langsrud, O. 2007. Fisher's exact test. http://www.matforsk.no/ola/fisher.htm

Leehey MA, Berry-Kravis E, Goetz CG, Zhang L, Hall DA, Li L, Rice CD, Lara R, Cogswell J, Reynolds A, Gane L, Jacquemont S, Tassone F, Grigsby J, Hagerman RJ, Hagerman PJ. 2007. FMR1 CGG repeat length predicts motor

dysfunction in premutation carriers. Neurology. 2007 Dec 5; [Epub ahead of print]

Lieber MR. 1998. Warner-Lambert/Parke-Davis Award Lecture. Pathological and physiological double-strand breaks: roles in cancer, aging, and the immune system. Am J Pathol 153:1323-1332.

Lilley DM. 1980. The inverted repeat as a recognizable structural feature in supercoiled DNA. Proc Natl Acad Sci USA 77:6468-6472.

Lobry, J.R. (1996) A simple vectorial representation of DNA sequences for the detection of replication origins in bacteria. Biochimie, 78, 323-326.

Lobry, J.R. (1999) Genomic landscapes. Microbiology Today, 26, 164-165.

Loots GG, Ovcharenko I. 2004. rVISTA 2.0: evolutionary analysis of transcription factor binding sites. Nucleic Acids Res. 32(Web Server issue):W217-21.

Lucier JF, Perreault J, Noël JF, Boire G, Perreault JP. 2007. RTAnalyzer: a web application for finding new retrotransposons and detecting L1 retrotransposition signatures. Nucleic Acids Res. 2007 Jul;35(Web Server issue):W269-74. Epub 2007 Jun 1.

Mrázek J, Xie S. 2006. Pattern locator: a new tool for finding local sequence patterns in genomic DNA sequences. Bioinformatics. 2006 Dec 15;22(24):3099-100. Epub 2006 Nov 8.

McMurray CT. 1995. Mechanisms of DNA expansion. Chromosoma. 104(1):2-13.

Mizraji E and Ninio J. 1985. Graphical coding of nucleic acid sequences. Biochimie. 67:445-448.

Mizuuchi K, Mizuuchi M, Gellert M. 1982. Cruciform structures in palindromic DNA are favored by DNA supercoiling. J Mol Biol 156:229-243.

Mukai JI. 1965. Species-specificity of nucleotide composition of ribonucleic acids from different silkworms. J Insect Physiol. 11:443-7.

Nakashima H, Ota M, Nishikawa K, Ooi T. 1998. Genes from nine genomes are separated into their organisms in the dinucleotide composition space. DNA Res. 5(5):251-9.

Nussinov R. 1987. Nucleotide quartets in the vicinity of eukaryotic transcriptional initiation sites: some DNA and chromatin structural implications. DNA. 6(1):13-22.

Nussinov R. 1984. Doublet frequencies in evolutionary distinct groups. Nucleic Acids Res. 1984 Feb 10;12(3):1749-63.

Nussinov R. 1983. Efficient algorithms for searching for exact repetition of nucleotide sequences. J Mol Evol. 1983;19(3-4):283-5.

Nussinov R. 1981. Nearest neighbor nucleotide patterns. Structural and biological implications. J Biol Chem. 256(16):8458-62.

Nussinov R, Jacobson AB. 1980. Fast algorithm for predicting the secondary structure of single-stranded RNA. Proc Natl Acad Sci U S A. 77(11):6309-13.

Okada N, Hamada M, Ogiwara I, Ohshima K. 1997. SINEs and LINEs share common 3' sequences: a review. Gene. 205(1-2):229-43.

Okada N, Ohshima K. 1995. Evolution of tRNA-derived SINEs *in* Maraia RJ, editor, *The impact of short interspersed elements (SINEs) on the host genome.* Austin: RG Landes Co., p 62-79.

Osawa S. 1960. The nucleotide composition of ribonucleic acids from subcellular components of yeast, Escherichia coli and rat liver, with special reference to the occurrence of pseudouridylic acid in soluble ribonucleic acid. Biochim. Biophys. Acta 42:244-254.

Panayotatos N, Wells RD. 1981. Cruciform structures in supercoiled DNA. Nature (London) 289:466-470.

Pandolfo M. 1998. Molecular genetics and pathogenesis of Friedreich ataxia. Neuromuscul. Disord 8:409-415.

Papatsenko D. 2007. ClusterDraw web server: a tool to identify and visualize clusters of binding motifs for transcription factors. Bioinformatics. 23(8):1032-4.

Park SJ, Wireman J, Summers AO. 1992. Genetic analysis of the Tn21 mer operator-promoter. J Bacteriol. 174(7):2160-71.

Paulson HL, Fischbeck. 1996. Trinucleotide repeats in neurogenetic disorders. Annu Rev Neurosci 19:79-107.

Pearson CE, Sinden RR. 1996. Alternative structures in duplex DNA formed within the trinucleotide repeats of the myotonic dystrophy and fragile X loci. Biochemistry. 35(15):5041-53.

Petruska J, Arnheim N, Goodman MF. 1996. Stability of intrastrand hairpin structures formed by the CAG/CTG class of DNA triplet repeats associated with neurological diseases. Nucleic Acids Res. 24(11):1992-8.

Platt, JR. 1955. Possible separation of inter-twisted nucleic acid chains by transfer-twist. Proc. Natl. Acad. Sci. USA 71, 181-183.

Portes-Sentis S, Sergeant A, Gruffat H. 1997. A particular DNA structure is required for the function of a *cis*-acting component of the Epstein-Barr virus OriLyt origin of replication. Nucleic Acids Res. 25(7):1347-1354.

Pugalenthi G, Suganthan PN, Sowdhamini R, Chakrabarti S. 2007. SMotif: a server for structural motifs in proteins. Bioinformatics. 23(5):637-8. Epub 2007 Jan 19.

Puntervoll P, Linding R, Gemünd C, Chabanis-Davidson S, Mattingsdal M, Cameron S, Martin DM, Ausiello G, Brannetti B, Costantini A, Ferrè F, Maselli V, Via A, Cesareni G, Diella F, Superti-Furga G, Wyrwicz L, Ramu C, McGuigan C, Gudavalli R, Letunic I, Bork P, Rychlewski L, Küster B, Helmer-Citterich M, Hunter WN, Aasland R, Gibson TJ. 2003. ELM server: a new resource for investigating short functional sites in modular eukaryotic proteins. Nucleic Acids Res., 31: 3625-3630.

Rao JE, Miller Ps, Craig NL. 2000. Recognition of triple-helical DNA structures by transposon Tn7. Saunderson RB, Yu B, Trent RJ, Pamphlett R. 2007. A comparison of the lengths of androgen receptor triplet repeats in brain and blood in motor neuron diseases. Proc Nat Acad Sci USA 97(8):3936-3941.

Roten CA, Gamba P, Barblan JL, Karamata D. 2002. Comparative Genometrics (CG): a database dedicated to biometric comparisons of whole genomes. Nucleic Acids Res 30(1):142-4.

Schattner P. 2002. Searching for RNA genes using base-composition statistics. Nucleic Acids Res. 30(9):2076-82.

Schnittman SM, Lane HC, Roth J, Burrows A, Folks TM, Kehrl JH, Koenig S, Berman P, Fauci AS. 1988. Characterization of GP120 binding to CD4 and an assay that measures ability of sera to inhibit this binding. J Immunol. 141(12):4181-6.

Schoonjans K, Staels B, Auwerx J. 1996. Role of the peroxisome proliferator-activated receptor (PPAR) in mediating the effects of fibrates and fatty acids on gene expression. J Lipid Res. 37(5):907-25.

Schroth GP and Ho PS. 1995. Occurrence of potential cruciform and H-DNA forming sequences in genomic DNA. Nucl Acids Res 23(11):1977-1983.

Shaffer PL, Gewirth DT. 2004. Vitamin D receptor-DNA interactions.

Vitam Horm. 68:257-73.

Shao J, Diamond MI. 2007. Polyglutamine diseases: emerging concepts in pathogenesis and therapy. Hum Mol Genet. 2007 Oct 15;16 Spec No. 2:R115-23.

Sharp PM, Bailes E, Robertson DL, Gao F, Hahn BH. 1999. Origins and Evolution of AIDS Viruses. Biol Bull 196:338-342.

She Q, Brügger K, Chen L. 2002. Archaeal integrative genetic elements and their impact on genome evolution. Res Microbiol. 153(6):325-32.

Shedlock AM, Okada N. 2000. SINE insertions: powerful tools for molecular systematics. Bioessays. 22(2):148-60.

Sia EA, Jinks-Robertson S, Peters TD. 1997. Genetic control of microsatellite instability. Mutat Res 383:62-70.

Sinden, RR. 1994. *DNA structure and function.* Academic Press, San Diego, CA, 398 pp.

Singer M and Berg P. 1991. *Genes and genomes.* Mill Valley, CA: University Science Books. 929 pp.

Sokol D, Benson G, Tojeira J. 2007. Tandem repeats over the edit distance. Bioinformatics. 23(2):e30-5.

Szklarczyk R, Heringa J. 2004. Tracking repeats using significance and transitivity. Bioinformatics. 20 Suppl 1:i311-7.

Szklarczyk R, Heringa J. 2006. AuberGene--a sensitive genome alignment tool. Bioinformatics. 22(12):1431-6.

Thomas JM, Horspool D, Brown G, Tcherepanov V, Upton C. 2007. GraphDNA: a Java program for graphical display of DNA composition analyses. BMC Bioinformatics 2007, 8:21.

Thompson JD, Higgins DG, Gibson TJ. 1994. CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. Nucleic Acids Res. 22:4673-80.

Tribioli C, Tamanini F, Patrosso C, Milanesi L, Villa A, Pergolizzi R, Maestrini E, Rivella S, Bione S, Mancini M *et al*, 1992. Methylation and sequence analysis around EagI sites: identification of 28 new CpG islands in XQ24-XQ28. Nucleic Acids Res. 20(4):727-33.

Ullu E, Tschudi C. 1984. Alu sequences are processed 7SL RNA genes. Nature 312:171-172.

Verrijdt G, Haelens A, Claessens F. 2003. Selective DNA recognition by the androgen receptor as a mechanism for hormone-specific regulation of gene expression. Mol Genet Metab. 2003 Mar;78(3):175-85.

Vansant G and Reynolds W. 1995. The consensus sequence of a major *Alu* subfamily contains a functional retinoic acid response element. Proc Natl Acad Sci USA, 92:8229-8233.

Watson JD and Crick FH. 1953. The structure of DNA. Cold Spring Harb Symp Quant Biol. 1953;18:123-31.

White HB 3rd, Laux BE, Dennis D. 1972. Messenger RNA structure: compatibility of hairpin loops with protein sequence. Science. 175(27):1264-6.

Woodworth-Gutai M, Lebowitz J. 1976. Introduction of interrupted secondary structure in supercoiled DNA as a function of superhelix density: Consideration of hairpin structures in superhelical DNA. J Virol 18:195-204.

Young ET, Sloan JS, van Riper K. 1999. Trinucleotide repeats are clustered in regulatory genes in *Saccharomyces cerevisiae*. Genetics 154:1053-1068.

Zhang F, Zhao Z. 2004. The influence of neighboring-nucleotide composition on single nucleotide polymorphisms (SNPs) in the mouse genome and its comparison with human SNPs. Genomics 84(5):785-95.

Zuker M. Mfold web server for nucleic acid folding and hybridization prediction. Nucleic Acids Res. 31 (13), 3406-3415, (2003)

12. Appendix

A. List of Publications (An asterick indicates appended papers)

*Lang, DM. 2007. Circuit Assemblages Derived From Net Dinucleotide Values Provide A Succinct Identity For The HIV-1 Genome And Each Of Its Genes. Virus Genes. PMID:17987374. **APPENDED.**

*Lang, DM. 2007. Imperfect DNA mirror repeats in the gag gene of HIV-1 (HXB2) coincide with protein structural elements and functional domains in each of the cleaved mature proteins. Virology J. 4(1):113. PMID:17963512. **APPENDED.**

Garcia E, Worsham P, Bearden S, Malfatti S, Lang D, Larimer F, Lindler L, Chain P. 2007. Pestoides F, an atypical Yersinia pestis strain from the former Soviet Union. Adv Exp Med Biol. 2007;603:17-22.

Reinis M, Weiser B, Kuiken C, Dong T, Lang D, Nachman S, Zhang Y, Rowland-Jones S, Burger H. 2007. Genomic Analysis of HIV Type 1 Strains Derived from a Mother and Child Pair of Long-Term Nonprogressors. AIDS Res Hum Retroviruses 23(2):309-15.

Gnanakaran S, Lang D, Daniels M, Bhattacharya T, Derdeyn CA, Korber B. 2006. Clade Specific Differences in HIV-1: Diversity and Correlations in C3-V4 Regions of gp120. J Virol. Dec 13; [Epub ahead of print].

Chohan B, Lang D, Sagar M, Korber B, Lavreys L, Richardson B, Overbaugh J. 2005. Selection for human immunodeficiency virus type 1 envelope glycosylation variants with shorter V1-V2 loop sequences occurs during transmission of certain genetic subtypes and may impact viral RNA levels. J Virol. 79(10):6528-31.

*Lang DM. 2005. Imperfect DNA mirror repeats in *E. coli* TnsA and other protein-coding DNA. 2005. Biosystems. 81(3):183-207. **APPENDED.**

Chohan B, Lang D, Sagar M, Korber B, Lavreys L, Richardson B and Overbaugh J. 2005. Selection for human immunodeficiency virus type 1 envelope glycosylation variants with shorter v1-v2 loop sequences occurs during transmission of certain genetic subtypes and may impact viral RNA levels. 2005. Jour. Virol. 79(10).

Fang G, Kuiken C, Weiser B, Rowland-Jones S, Plummer F, Chen CH, Kaul R, Anzala AO, Bwayo J, Kimani J, Philpott SM, Kitchen C, Sinsheimer JS, Gaschen B, Lang D, Shi B, Kemal KS, Rostron T, Brunner C, Beddows S, Sattenau Q, Paxinos E, Oyugi J, Burger H. 2004. Long-term survivors in Nairobi: complete HIV-1 RNA sequences and immunogenetic associations. J Infect Dis. 190(4):697-701.

Fan X, Lang DM, Xu Y, Lyra AC, Yusim K, Everhart JE, Korber BT, Perelson AS, Di Bisceglie AM. 2003. Liver transplantation with hepatitis C virus-infected graft: interaction between donor and recipient viral strains. Hepatology. Jul;38(1):25-33.

Lyra AC, Fan X, Lang DM, Yusim K, Ramrakhiani S, Brunt EM, Korber B, Perelson AS, Di Bisceglie AM. 2002. Evolution of hepatitis C viral quasispecies after liver transplantation. Gastroenterology. Nov;123(5):1485-93.

Gaschen B, Taylor J, Yusim K, Foley B, Gao F, Lang D, Novitsky V, Haynes B, Hahn BH, Bhattacharya T, Korber B. 2002. Diversity considerations in HIV-1 vaccine selection. Science. Jun 28;296(5577):2354-60.

Krogstad P, Eshleman SH, Geng Y, Jackson JB, Wantman M, Korber B, Lang D, Wiznia A, Johnson G, Nachman S, Palumbo P. 2002. Mother-to-child transmission in the United States of subtypes D and A/G human immunodeficiency virus type 1. AIDS Res Hum Retroviruses.18:413-7.

Thakallapally R, Kibbe W, Lang D, Korber B. 2000. Motifscan: A Weβ–based Tool to Find HLA Anchor Residues in Proteins or Peptides. pp. I-101-102 in HIV Molecular Immunology Database 2000. Edited by: Korber BT, Brander C, Haynes BF, Koup R, Kuiken C, Moore JP, Walker BD, and Watkins D. Published by: Theoretical Biology and Biophysics Group, Los Alamos National Laboratory, Los Alamos, NM.

*Lang DM. 2000. Net nearest neighbor analysis (NNNA) summarizes non-compensated dinucleotides within gene sequences. Bioinformatics. Mar;16(3):212-21. **APPENDED.**

HIV Molecular Immunology Database 2002, 2001, 2000. Editors: Bette T. M. Korber, Christian Brander, Barton F. Haynes, Richard Koup, Carla Kuiken, John P. Moore, Bruce D. Walker, and David I. Watkins. Published by: Theoretical Biology and Biophysics Group, Los Alamos National Laboratory, Theoretical Biology and Biophysics, Los Alamos, NM.

HIV Sequence Compendium 2002, 2001, 2000. Editors: Carla Kuiken, Brian Foley, Eric Freed, Beatrice Hahn, Preston Marx, Francine McCutchan, John W. Mellors, Steven Wolinsky, Bette Korber. Publisher: Theoretical Biology and Biophysics Group, Los Alamos National Laboratory, Theoretical Biology and Biophysics, Los Alamos, NM.

B. Perl programs

i. Perl programs overview

The perl programs in the following chapters include 3 programs related to the conversion of DNA sequences into DNA circuits, and 3 programs related to the identification of mIMRs and rdIMRs.

In order to determine circuits, the program <count_nucleotides_dinucleotides_net-Dinucleotides> (B.ii.) will take a file consisting of multiple fasta sequences and count the indicated values for each sequence. The output of this file is input into <convert_negative_net_dinucleotides> (B.iii), which converts any negative net dinucletides into positive values. Positive values are required by the program that calculates circuits. The program <convert_net_dinucleotides_to_circuits> (B.iv) uses the output of <convert_negative_net_dinucleotides> to compute the type and number of circuits for each sequence.

The determination of IMRs involves identifying both mIMRs (repeats that were found to span protein domains and contain multiple PSEs) and rdIMRs (repeats that were found to span a single PSE). The program <find_max_mirror_repeats> (B.v) identifies all mIMRs in a sequence. The program <find_rd_mirror_repeats> (B.vi) identifies all rdIMRs in a sequence. Both category of repeats are each (separately) evaluated to identify their hierarchal structure using the program <nest_repeats> (B.vii).

The <find_max_mirror_repeats> program evaluates all IMRs contained within the sequence by progressively evaluating the symmetry of each possible string within the sequence. This process identifies the largest IMRs in the sequence. The <find_rd_mirror _repeats> program evaluates the sequence sequentially, beginning at the 5'-start position. It identifies a dinucleotide and its nearest downstream reverse dinucleotide (e.g. AC and CA), and the symmetry of the sequence within this span.

The nest repeats program identifies the hierarchal structure of IMRs by sequentially evaluating the output of the <find_max_mirror_repeats> and <find_rd_mirror_repeats>

programs, from the 5-position start of the DNA to the 3-position end. IMRs that uniquely span a particular region are identified as L1 IMRs and those IMRs that are contained within L1 IMRs are assigned to higher levels (L2, L3, etc). The next downstream L1 IMR ends beyond the span of the previous L1 IMR; it may begin within the previous IMR or after its end.

ii. Perl program. count_nucleotides_dinucleotides_netDinucleotides

```
#!/usr/bin/perl
#use strict;

print "Start\n\n";

# any number of fasta files can be pasted within the array framework, i.e.
# @list=qw(paste your fasta files here);
# HOWEVER, the header line may not contain any blanks, and should end with #
# HOWEVER, the sequence line must be a single line, no end of line characters
# THE RESULTS of this file will be copied from the screen, and for additional
# processing, be pasted into the Perl program <convert_negative_net_dinucleotides>

@list=qw(
>gi|51594359:509245-509550_TB32953#
ATGGCTAAGGGGCAATCTTTGCAAGATCCGTTCCTGAACGCATTGCGTCGTGAACGGGTTCCGGTTTCTATTTATTTAGTGAATGGTATTA
AACTGCAGGGCCAAGTTGAGTCTTTTGATCAGTTTGTCATTCTGTTAAAAAATACAGTCAGCCAGATGGTTTATAAGCACGCCATCTCTAC
TGTTGTGCCTTCTCGTCCGGTTTCGCATCACAGCAATACTCCGAGCGGTAGCACCAATAATTATCATGGTAGTAATCCGTCTGCGCCGCAA
CAGCCGCAGCAGGACAGTGATGACGCTGAATAA
);
#print "@list";
print "name length NetAC NetAG NetAT NetCG NetCT NetGT sumGG sumCC sumTT sumAA \n";

for ($i=0; $i < @list; $i=$i+1) {
    print ">$list[$i] ";
    #print "$list[$i+1], ";
    $i=$i+1;
    countdinucs();
    #print "\n\n";
}
print "End\n";

sub countdinucs {

    # Put string into an array.
    $list[$i] = lc($list[$i]);
    @DNA = split( '', $list[$i] );

    # Initialize the counts.
    $sumAA=0; $sumAC=0; $sumAG=0; $sumAT=0;
    $sumCA=0; $sumCC=0; $sumCG=0; $sumCT=0;
    $sumGA=0; $sumGC=0; $sumGG=0; $sumGT=0;
    $sumTA=0; $sumTC=0; $sumTG=0; $sumTT=0;
    $sumA=0; $sumC=0; $sumG=0; $sumT=0;
    $NetAC=0; $NetAG=0; $NetAT=0; $NetCG=0; $NetCT=0; $NetGT=0;

    # In loop, count each type of nucleotide.
    for ($j=0; $j< @DNA; $j=$j+1) {

        if    ($DNA[$j] eq "a") { $sumA= $sumA + 1;}
        elsif ($DNA[$j] eq "c") { $sumC= $sumC + 1;}
        elsif ($DNA[$j] eq "g") { $sumG= $sumG + 1;}
        elsif ($DNA[$j] eq "t") { $sumT= $sumT + 1;}
```

```
        if    (($DNA[$j] eq "a") and ($DNA[$j+1] eq "a")) {$sumAA= $sumAA + 1;}
        elsif (($DNA[$j] eq "a") and ($DNA[$j+1] eq "c")) {$sumAC= $sumAC + 1;}
        elsif (($DNA[$j] eq "a") and ($DNA[$j+1] eq "g")) {$sumAG= $sumAG + 1;}
        elsif (($DNA[$j] eq "a") and ($DNA[$j+1] eq "t")) {$sumAT= $sumAT + 1;}
        elsif (($DNA[$j] eq "c") and ($DNA[$j+1] eq "a")) {$sumCA= $sumCA + 1;}
        elsif (($DNA[$j] eq "c") and ($DNA[$j+1] eq "c")) {$sumCC= $sumCC + 1;}
        elsif (($DNA[$j] eq "c") and ($DNA[$j+1] eq "g")) {$sumCG= $sumCG + 1;}
        elsif (($DNA[$j] eq "c") and ($DNA[$j+1] eq "t")) {$sumCT= $sumCT + 1;}
        elsif (($DNA[$j] eq "g") and ($DNA[$j+1] eq "a")) {$sumGA= $sumGA + 1;}
        elsif (($DNA[$j] eq "g") and ($DNA[$j+1] eq "c")) {$sumGC= $sumGC + 1;}
        elsif (($DNA[$j] eq "g") and ($DNA[$j+1] eq "g")) {$sumGG= $sumGG + 1;}
        elsif (($DNA[$j] eq "g") and ($DNA[$j+1] eq "t")) {$sumGT= $sumGT + 1;}
        elsif (($DNA[$j] eq "t") and ($DNA[$j+1] eq "a")) {$sumTA= $sumTA + 1;}
        elsif (($DNA[$j] eq "t") and ($DNA[$j+1] eq "c")) {$sumTC= $sumTC + 1;}
        elsif (($DNA[$j] eq "t") and ($DNA[$j+1] eq "g")) {$sumTG= $sumTG + 1;}
        elsif (($DNA[$j] eq "t") and ($DNA[$j+1] eq "t")) {$sumTT= $sumTT + 1;}
    }

    $length = @DNA;
    #print "Sequence length =SL
    $NetAC = $sumAC - $sumCA; $NetAG = $sumAG - $sumGA; $NetAT = $sumAT - $sumTA;
    $NetCG = $sumCG - $sumGC; $NetCT = $sumCT - $sumTC; $NetGT = $sumGT - $sumTG;


    #print "$length $sumA $sumC $sumG $sumT $sumAA $sumAC $sumAG $sumAT $sumCA $sumCC
    #$sumCG $sumCT $sumGA $sumGC $sumGG $sumGT $sumTA $sumTC $sumTG $sumTT $NetAC #$NetAG
    $NetAT $NetCG $NetCT $NetGT $sumGG $sumCC $sumTT $sumAA \n";

print "$length $NetAC $NetAG $NetAT $NetCG $NetCT $NetGT $sumGG $sumCC $sumTT $sumAA \n";
}
```

iii. Perl program. convert_negative_net_dinucleotides

This program is a required step to create the input for the program that is used to create circuits from net dinucleotide values (iv. Perl program. convert_net_dinucleotides_ to_circuits). The convert_negative_net_dinucleotides program uses the screen output from the previous program (13. Perl program. count_nucleotides_dinucleotides_ netDinucleotides) and converts net dinucleotides that have a negative value into positive values. For example nAT = -1 equals nTA = 1, and nGA = -3 equals nAG=3.

```
#!/usr/bin/perl

# this program makes all net dinucs positive
# it uses output from net dinucleotide program (#13) which is tab delimited net dinucs

# REQUIRED FORMAT FOR THIS PROGRAM (Output of #13)
# >CPZ.US.85.CPZUS_AF103818#
# 3225 -10 3 7 -9 -1 -6 29 7 11 35
# >CPZ.GA..CPZGAB_X52154#
# 207 -6 6 0 -10 4 -4 25 9 12 36

# REQUIRED. no spaces in header line
# REQUIRED. # at end of header line
# be sure input data is separated by spaces
# PASTE output from #13 within @convert array
# @convert = qw(paste #13 output here);


@convert = qw(
>CPZ.US.85.CPZUS_AF103818#
225 -10 3 7 -9 -1 -6 29 7 11 35
>CPZ.GA..CPZGAB_X52154#
```

```
207 -6 6 0 -10 4 -4 25 9 12 36
);

$length_conv=((@convert/11)-1);

for ($c=0;$c<$length_conv;$c++) {

    for ($i=0;$i<$length_conv;$i++) {

    #_____________________________________________
    #seq_name seq_length

            $seq_name[$i] = $convert[$c]."#"; $c++;
            #print "i $i seq_name $seq_name[$i]\n";
            $seq_length[$i] = $convert[$c]; $c++;
            #print "i $i seq_length $seq_length[$i] \n";

     #_____________________________________________
    #convert ±$AC -> $AC, $mA

            $AC_test = $convert[$c];
            if ($AC_test == 0) { #print "equal to\n";
                $mAC[$i] = 0; $AC[$i] = 0; $c++;
                #print "i $i AC $AC[$i] mAC $mAC[$i] \n\n";
            }
            if ($AC_test > 0) { #print "greater than\n";
                $AC[$i] = $convert[$c]; $mAC[$i] = 0; $c++;
                #print "i $i AC $AC[$i] mAC $mAC[$i] \n\n";
            }
            if ($AC_test < 0) { #print "less than\n";
                $mAC[$i] = abs($convert[$c]); $AC[$i] = 0; $c++;
                #print "i $i AC $AC[$i] mAC $mAC[$i] \n\n";
            }

     #_____________________________________________
    #convert ±$AG -> $AG, $mAG

            $AG_test = $convert[$c];
            if ($AG_test == 0) { #print "equal to\n";
                $mAG[$i] = 0; $AG[$i] = 0; $c++;
            }
                if ($AG_test > 0) { #print "greater than\n";
                $AG[$i] = $convert[$c]; $mAG[$i] = 0; $c++;
            }
            if ($AG_test < 0) { #print "less than\n";
                $mAG[$i] = abs($convert[$c]); $AG[$i] = 0; $c++;
                #print "i $i AG $AG[$i] mAG $mAG[$i] \n\n";
            }

     #_____________________________________________
    #convert ±$AT -> $At, $mAT
            $AT_test = $convert[$c];
            if ($AT_test == 0) {#print "equal to\n";
                $mAT[$i] = 0; $AT[$i] = 0; $c++;
                #print "i $i AT $AT[$i] mAT $mAT[$i] \n\n";
            }
            if ($AT_test > 0) { #print "greater than\n";
                $AT[$i] = $convert[$c]; $mAT[$i] = 0; $c++;
                #print "i $i AT $AT[$i] mAT $mAT[$i] \n\n";
            }
            if ($AT_test < 0) { #print "less than\n";
                $mAT[$i] = abs($convert[$c]); $AT[$i] = 0; $c++;
                #print "i $i AT $AT[$i] mAT $mAT[$i] \n\n";
            }

     #_____________________________________________
    #convert ±$CG -> $CG, $mCG

            $CG_test = $convert[$c];
            if ($CG_test == 0) { #print "equal to\n";
                $mCG[$i] = 0; $CG[$i] = 0; $c++;
                #print "i $i CG $CG[$i] mCG $mCG[$i] \n\n";
```

```
            }
            if ($CG_test > 0) { #print "greater than\n";
                $CG[$i] = $convert[$c]; $mCG[$i] = 0; $c++;
                #print "i $i CG $CG[$i] mAG $mCG[$i] \n\n";
            }
            if ($CG_test < 0) { #print "less than\n";
                $mCG[$i] = abs($convert[$c]); $CG[$i] = 0; $c++;
                #print "i $i CG $CG[$i] mCG $mCG[$i] \n\n";
            }
            #_____________________________________________
    #convert ±$CT -> $CT, $mCT

            $CT_test = $convert[$c];
            if ($CT_test == 0) { #print "equal to\n";
                $mCT[$i] = 0; $CT[$i] = 0; $c++;
                #print "i $i CT $CT[$i] mCT $mCT[$i] \n\n";
            }
            if ($CT_test > 0) { #print "greater than\n";
                $CT[$i] = $convert[$c]; $mCT[$i] = 0; $c++;
                #print "i $i CT $CT[$i] mAG $mCT[$i] \n\n";
            }
            if ($CT_test < 0) { #print "less than\n";
                $mCT[$i] = abs($convert[$c]); $CT[$i] = 0; $c++;
                #print "i $i CT $CT[$i] mCT $mCT[$i] \n\n";
            }
    #_____________________________________________
    #convert ±$GT -> $GT, $mGT

            $GT_test = $convert[$c];
            if ($GT_test == 0) { #print "equal to\n";
                $mGT[$i] = 0; $GT[$i] = 0; $c++;
                #print "i $i GT $GT[$i] mGT $mGT[$i] \n\n";
            }
            if ($GT_test > 0) { #print "greater than\n";
                $GT[$i] = $convert[$c]; $mGT[$i] = 0; $c++;
                #print "i $i GT $GT[$i] mAG $mGT[$i] \n\n";
            }
            if ($GT_test < 0) { #print "less than\n";
                $mGT[$i] = abs($convert[$c]); $GT[$i] = 0; $c++;
                #print "i $i GT $GT[$i] mGT $mGT[$i] \n\n";
            }
    #_____________________________________________

            #print GG, CC, TT, AA
            #print "GG i $i convert[$c] $convert[$c] \n";
            $GG[$i] = $convert[$c]; $c++;

            #print "CC i $i convert[$c] $convert[$c] \n";
            $CC[$i] = $convert[$c]; $c++;

            #print "TT i $i convert[$c] $convert[$c] \n";
            $TT[$i] = $convert[$c]; $c++;

            #print "AA i $i convert[$c] $convert[$c] \n";
            $AA[$i] = $convert[$c]; $c++;

}   }



    print "nm l GG CC TT AA AC mAC AG mAG AT mAT CG mCG CT mCT GT mGT \n";


for ($i=0;$i<$length_conv;$i++) {
    print "$seq_name[$i] $seq_length[$i] $GG[$i] $CC[$i] $TT[$i] $AA[$i] $AC[$i] $mAC[$i]
    $AG[$i] $mAG[$i] $AT[$i] $mAT[$i] $CG[$i] $mCG[$i] $CT[$i] $mCT[$i] $GT[$i] $mGT[$i]
    \n";
}
```

iv. Perl program. convert_net_dinucleotides_to_circuits

```
# 30jan2006 checked for tags with GATCG
# AC=0, CG=4, GT=4, TA=1, TC=4 has 2 solutions
# 2 solution problems not yet resolvable, but rare

=pod
Specific directions for this program begin here and end a "=cut"
The program itself begins after "=cut".
FORMAT FOR THIS PROGRAM REQUIRES MANUAL INTERVENTION
THESE INTERVENTIONS SHOULD OPERATE ONLY ON THE INPUT FILE

Start with the output of the program <convert_negative_net_dinucleotides> illustrated
below.

>CPZ.US.85.CPZUS_AF103818# 225 29 7 11 35 0 10 3 0 7 0 0 9 0 1 0 6
>CPZ.GA..CPZGAB_X52154# 207 25 9 12 36 0 6 6 0 0 0 0 10 4 0 0 4

SUBSTITUTE. for <# > substitute <#^p>
This results in the following input.

>CPZ.US.85.CPZUS_AF103818#
225 29 7 11 35 0 10 3 0 7 0 0 9 0 1 0 6
>CPZ.GA..CPZGAB_X52154#
207 25 9 12 36 0 6 6 0 0 0 0 10 4 0 0 4

SUBSTITUTE. for < > substitute <xxx>
This substitutes xxx for blank spaces, and results in the following input.

>>a1.FI.91.FIN91121_aF219261#
0xxx3xxx1xxx0xxxxxx3xxx0xxx0xxx7xxx4xxx0xxx0xxx7
>>a1.FI.91.FIN9199_aF219265#
0xxx2xxx4xxx0xxxxxx0xxx1xxx0xxx7xxx5xxx0xxx0xxx4
>>a1.KE.00.KNH1214_aF457071#
0xxx3xxx4xxx0xxxxxx0xxx0xxx0xxx9xxx6xxx0xxx0xxx6

AC mAC AG  mAG  AT mAT  CG mCG  CT  mCT GT  mGT
=cut


print "Start program\n";
print "name length GG CC TT AA GCATG GTACG GACTG GTCAG GATCG GCTAG GCAG ACTA GTAG GTCG GACG
ATCA GCTG GATG TAG \n";

$GG=0; $CC=0; $TT=0; $AA=0; $GCATG=0; $GTACG=0; $GACTG=0; $GTCAG=0; $GATCG=0; $GCTAG=0;
$GCAG=0; $ACTA=0;
$GTAG=0; $GTCG=0; $GACG=0; $ATCA=0; $GCTG=0; $GATG=0; $tag=0;
$AC=0; $mAC=0; $AG=0; $mAG=0; $AT=0; $mAT=0;
$CG=0; $mCG=0; $CT=0; $mCT=0; $GT=0; $mGT=0;

#Enter your data here
@list = qw(>>gi|51594359:509245-509550_TB32953#
306xxx15xxx13xxx27xxx22xxx0xxx12xxx7xxx0xxx5xxx0xxx0xxx5xxx0xxx7xxx2xxx0);

# 1  2   3   4   5   6   7   8    9  10   11  12
# AC mAC AG  mAG  AT mAT  CG mCG  CT  mCT GT  mGT

$l_list = @list;
for ($i=0;$i<$l_list;$i++) {
    $i++;
    circuits();
}


sub circuits {

($length,$GG,$CC,$TT,$AA,$AC,$mAC,$AG,$mAG,$AT,$mAT,$CG,$mCG,$CT,$mCT,$GT,$mGT)  =
split(/xxx/,$list[$i]);

#_____
```

```
# GCATG   mCG mAC AT mGT #
#_____________________________________________

if (($mCG<=$mAC)&&($mCG<=$AT)&&($mCG<=$mGT)&&($mCG>0)&&($mAC>0)&&($AT>0)&&($mGT>0)) {
        ##print "mCG least \n";
        $GCATG = $mCG;
}
if
(($mAC<=$mCG)&&($mAC<=$AT)&&($mAC<=$mGT)&&($GCATG==0)&&
($mCG>0)&&($mAC>0)&&($AT>0)&&($mGT>0)) {
        ##print "mAC least \n";
        $GCATG = $mAC;
}
if (($AT <= $mAC)&&($AT <=$mCG)&&($AT<=$mGT) && ($GCATG == 0)
&&($mCG>0)&&($mAC>0)&&($AT>0)&&($mGT>0)) {
        ##print "AT least \n";
        $GCATG = $AT;
}
if (($mGT <= $mAC)&&($AT <=$mCG)&&($mGT<=$AT) && ($GCATG == 0)
&&($mCG>0)&&($mAC>0)&&($AT>0)&&($mGT>0)) {
        ##print "mGT least \n";
        $GCATG = $mGT;
}
$mAC=$mAC-$GCATG; $mCG=$mCG-$GCATG; $mGT=$mGT-$GCATG; $AT=$AT-$GCATG;

#_____________________________________________
# GTACG   GT mAT AC CG #
#_____________________________________________

if (($GT <= $mAT) && ($GT <=$AC) && ($GT <= $CG) && ($GT>0)&&($mAT>0)&&($AC>0)&&($CG>0)) {
        ##print "GT least \n";
        $GTACG = $GT;
}
if (($mAT <= $GT)&&($mAT<= $AC)&&($mAT<=$CG)&&($GTACG == 0) &&
($GT>0)&&($mAT>0)&&($AC>0)&&($CG>0)) {
        ##print "mAT least \n";
        $GTACG = $mAT;
}
if (( $AC <= $GT)&&( $AC<=$mAT)&&( $AC<=$CG)&&($GTACG == 0) &&
($GT>0)&&($mAT>0)&&($AC>0)&&($CG>0)) {
        ##print "AC least \n";
        $GTACG = $AC;
}
if (( $CG <= $GT)&&( $CG<=$mAT)&&( $CG<=$AC)&&($GTACG == 0) &&
($GT>0)&&($mAT>0)&&($AC>0)&&($CG>0)) {
        ##print "CG least \n";
        $GTACG = $CG;
}
$GT=$GT-$GTACG; $mAT=$mAT-$GTACG; $AC=$AC-$GTACG; $CG=$CG-$GTACG;

#_____________________________________________
# GACTG   mAG AC CT mGT #
#_____________________________________________

if (($mAG <= $AC) && ($mAG <=$CT) && ($mAG <= $mGT) &&
($mAG>0)&&($AC>0)&&($CT>0)&&($mGT>0)) {
        ##print "mAG least \n";
        $GACTG = $mAG;
}
if (($AC <= $mAG)&&($AC<=$CT)&&($AC<=$mGT) && ($GACTG == 0) &&
($mAG>0)&&($AC>0)&&($CT>0)&&($mGT>0)) {
        ##print "AC least \n";
        $GACTG = $AC;
}
if (($CT <= $mAG)&&($CT <=$AC)&&($CT<=$mGT) && ($GACTG == 0) &&
($mAG>0)&&($AC>0)&&($CT>0)&&($mGT>0)) {
        ##print "CT least \n";
        $GACTG = $CT;
}
if (($mGT <= $mAG)&&($mGT <=$AC)&&($mGT<=$CT) && ($GACTG == 0) &&
($mAG>0)&&($AC>0)&&($CT>0)&&($mGT>0)) {
        ##print "mGT least \n";
```

```
        $GACTG = $mGT;
}
$mAG=$mAG-$GACTG; $AC=$AC-$GACTG; $CT=$CT-$GACTG; $mGT=$mGT-$GACTG;

#_____________________________________________
# GTCAG   GT mCT mAC AG #
#_____________________________________________

if (($GT <= $mCT) && ($GT <=$mAC) && ($GT <= $AG) &&($GT>0)&&($mCT>0)&&($mAC>0)&&($AG>0)) {
        ##print "GT least \n";
        $GTCAG = $GT;
}
if (($mCT <= $GT)&&($mCT<=$mAC)&&($mCT<=$mAG) && ($GTCAG == 0)
&&($GT>0)&&($mCT>0)&&($mAC>0)&&($AG>0)) {
        ##print "mCT least \n";
        $GTCAG = $mCT;
}
if (($mAC <= $GT)&&($mAC <=$mCT)&&($mAC<=$AG) && ($GTCAG == 0)
&&($GT>0)&&($mCT>0)&&($mAC>0)&&($AG>0)) {
        ##print "mAC least \n";
        $GTCAG = $mAC;
}
if (($AG <= $GT)&&($AG <=$mCT)&&($AG<=$mAC) && ($GTCAG == 0)
&&($GT>0)&&($mCT>0)&&($mAC>0)&&($AG>0)) {
        ##print "AG least \n";
        $GTCAG = $AG;
}
$GT=$GT-$GTCAG; $mCT=$mCT-$GTCAG; $mAC=$mAC-$GTCAG; $AG=$AG-$GTCAG;

#_____________________________________________
# GATCG   mAG AT mCT CG #
#_____________________________________________

if (($mAG <= $AT) && ($mAG <=$mCT) && ($mAG <= $CG) &&($mAG>0)&&($AT>0)&&($mCT>0)&&($CG>0))
{
        ##print "mAG least \n";
        $GATCG = $mAG;
}
if (($AT <= $mAG)&&($AT <=$mCT)&&($AT <=$CG) && ($GATCG == 0)
&&($mAG>0)&&($AT>0)&&($mCT>0)&&($CG>0)) {
        ##print "AT least \n";
        $GATCG = $AT;
}
if (($mCT <= $mAG)&&($mCT <=$AT)&&($mCT <=$CG) && ($GATCG == 0)
&&($mAG>0)&&($AT>0)&&($mCT>0)&&($CG>0)) {
        ##print "mCT least \n";
        $GATCG = $mCT;
}
if (($CG <= $mAG)&&($CG <=$AT)&&($CG<=$mCT) && ($GATCG == 0)
&&($mAG>0)&&($AT>0)&&($mCT>0)&&($CG>0)) {
        ##print "CG least \n";
        $GATCG = $CG;
}
$mAG=$mAG-$GATCG; $AT=$AT-$GATCG; $mCT=$mCT-$GATCG; $CG=$CG-$GATCG;

#_____________________________________________
# GCTAG   mCG CT mAT AG #
#_____________________________________________

if (($mCG <= $CT) && ($mCG <=$mAT) && ($mCG <= $AG) &&($mCG>0)&&($CT>0)&&($mAT>0)&&($AG>0))
{
        ##print "mCG least \n";
        $GCTAG = $mCG;
}
if (($CT <= $mCG)&&($CT<=$mAT)&&($CT<=$AG) && ($GCTAG == 0)
&&($mCG>0)&&($CT>0)&&($mAT>0)&&($AG>0)) {
        ##print "CT least \n";
        $GCTAG = $CT;
}
if (($mAT <= $mCG)&&($mAT <=$CT)&&($mAT<=$AG) && ($GCTAG == 0)
&&($mCG>0)&&($CT>0)&&($mAT>0)&&($AG>0)) {
        ##print "mAT least \n";
```

```
        $GCTAG = $mAT;
}
if (($AG <= $mCG)&&($AG <=$CT)&&($AG<=$mAT) && ($GCTAG == 0)
&&($mCG>0)&&($CT>0)&&($mAT>0)&&($AG>0)) {
        ##print "AG least \n";
        $GCTAG = $AG;
}
$mCG=$mCG-$GCTAG; $CT=$CT-$GCTAG; $mAT=$mAT-$GCTAG; $AG=$AG-$GCTAG;

#_____________________________________________________
# GCAG     mCG mAC AG  #
#_____________________________________________________

if (($mCG<=$mAC)&&($mCG<= $AG) &&($mCG>0)&&($mAC>0)&&($AG>0)) {$GCAG = $mCG;}
if (($mAC<=$mCG)&&($mAC<= $AG)&&($GCAG==0) &&($mCG>0)&&($mAC>0)&&($AG>0)){$GCAG=$mAC;}
if (( $AG<=$mAC)&&( $AG<=$mCG)&&($GCAG==0)     &&($mCG>0)&&($mAC>0)&&($AG>0)){$GCAG= $AG;}
$mAC=$mAC-$GCAG; $mCG=$mCG-$ GCAG; $AG=$AG-$GCAG;

#_____________________________________________________
# ACTA    AC CT mAT  #
#_____________________________________________________

if (( $AC<=$CT)&&( $AC<= $mAT)&&($AC>0)&&($CT>0)&&($mAT>0)) {$ACTA = $AC;}
if (( $CT<=$AC)&&( $CT<= $mAT)&&($ACTA==0)      &&($AC>0)&&($CT>0)&&($mAT>0)){$ACTA= $CT;}
if (($mAT<=$AC)&&($mAT<= $CT) &&($ACTA==0) &&($AC>0)&&($CT>0)&&($mAT>0)){$ACTA=$mAT;}
$AC=$AC-$ACTA; $CT=$CT-$ ACTA; $mAT=$mAT-$ACTA;

#_____________________________________________________
# GTAG     GT mAT AG  #
#_____________________________________________________

if (( $GT<=$mAT)&&( $GT<= $AG) &&($GT>0)&&($mAT>0)&&($AG>0)) {$GTAG = $GT;}
if (($mAT<= $GT)&&($mAT<= $AG)&&($GTAG==0) &&($GT>0)&&($mAT>0)&&($AG>0)){$GTAG=$mAT;}
if (( $AG<= $GT)&&( $AG<=$mCT)&&($GTAG==0)      &&($GT>0)&&($mAT>0)&&($AG>0)){$GTAG= $AG;}
$GT=$GT-$GTAG; $mAT=$mAT-$ GTAG; $AG=$AG-$GTAG;

#_____________________________________________________
# GTCG     GT mCT CG  #
#_____________________________________________________

if (( $GT<=$mCT)&&( $GT<=$CG) &&($GT>0)&&($mCT>0)&&($CG>0)) {$GTCG = $GT;}

if (($mCT<= $GT)&&($mCT<= $CG)&&($GTCG==0)&&($GT>0)&&($mCT>0)&&($CG>0)){$GTCG=$mCT;}

if (( $CG<=$GT)&&( $CG<=$mCT)&&($GTCG==0) &&($GT>0)&&($mCT>0)&&($CG>0)){$GTCG= $CG;}

$GT=$GT-$GTCG; $mCT=$mCT-$GTCG; $CG=$CG-$GTCG;

#_____________________________________________________
# GACG     mAG AC CG
#_____________________________________________________

if (($mAG<= $AC)&&($mAG<=$CG)&&($mAG>0)&&($AC>0)&&($CG>0)) {$GACG = $mAG;}
if (( $AC<=$mAG)&&( $AC<=$CG)&&($GACG==0) &&($mAG>0)&&($AC>0)&&($CG>0)){$GACG=$AC;}
if (( $CG<=$mAG)&&( $CG<=$AC)&&($GACG==0) &&($mAG>0)&&($AC>0)&&($CG>0)){$GACG=$CG;}
$mAG=$mAG-$GACG; $AC=$AC-$ GACG; $CG=$CG-$GACG;

#_____________________________________________________
# ATCA     AT mCT mAC  #
#_____________________________________________________

if (( $AT<=$mCT)&&( $AT<=$mAC) &&($AT>0)&&($mCT>0)&&($mAC>0))  {$ATCA = $AT;}
if (($mCT<= $AT)&&($mCT<=$mAC)&&($ATCA==0) &&($AT>0)&&($mCT>0)&&($mAC>0)){$ATCA=$mCT;}
if (($mAC<= $AT)&&($mAC<=$mCT)&&($ATCA==0)     &&($AT>0)&&($mCT>0)&&($mAC>0)){$ATCA= $mAC;}
$AT=$AT-$ATCA; $mCT=$mCT-$ ATCA; $mAC=$mAC-$ATCA;

#_____________________________________________________
# GCTG     mCG CT mGT  #
#_____________________________________________________
#

if (($mCG<= $CT)&&($mCG<=$mGT) &&($mCG>0)&&($CT>0)&&($mGT>0)) {$GCTG = $mCG;}
```

```
if (( $CT<=$mCG)&&( $CT<=$mGT)&&($GCTG==0)    &&($mCG>0)&&($CT>0)&&($mGT>0)){$GCTG= $CT;}
if (($mGT<=$mCG)&&($mGT<= $CT)&&($GCTG==0) &&($mCG>0)&&($CT>0)&&($mGT>0)){$GCTG=$mGT;}
$mCG=$mCG-$GCTG; $CT=$CT-$ GCTG; $mGT=$mGT-$GCTG;

#_____________________________________________
# GATG    mAG AT mGT
#_____________________________________________

if (($mAG<=$mAT)&&($mAG<= $mGT) &&($mAG>0)&&($AT>0)&&($mGT>0)) {$GATG = $mAG;}
if (( $AT<=$mAG)&&( $AT<= $mGT)&&($GATG==0)   &&($mAG>0)&&($AT>0)&&($mGT>0)){$GATG= $AT;}
if (($mGT<=$mAG)&&($mGT<=  $AT)&&($GATG==0) &&($mAG>0)&&($AT>0)&&($mGT>0)){$GATG=$mGT;}
$mAG=$mAG-$GATG; $AT=$AT-$ GATG; $mGT=$mGT-$GATG;

#_____________________________________________

if (($AC+$mAC+$AG+$mAG+$AT+$mAT+$CG+$mCG+$CT+$mCT+$GT+$mGT) < 0) {
print "$list[$i-1]# $length $GG $CC $TT $AA $GCATG $GTACG $GACTG $GTCAG $GATCG $GCTAG $GCAG
$ACTA $GTAG $GTCG $GACG $ATCA $GCTG $GATG TAG $tag check_values ";
       if ($AC>0) {print "AC $AC ";}
       if ($mAC>0) {print "CA $mAC ";}
       if ($AG>0) {print "AG $AG ";}
       if ($mAG>0) {print "GA $mAG ";}
       if ($AT>0) {print "AT $AT ";}
       if ($mAT>0) {print "TA $mAT ";}
       if ($CG>0) {print "CG $CG ";}
       if ($mCG >0) {print "GC $mCG ";}
       if ($CT>0) {print "CT $CT ";}
       if ($mCT >0) {print "TC $mCT ";}
       if ($GT>0) {print "GT $GT ";}
       if ($mGT >0) {print "TG $mGT ";}
     #print "\n";

#$sumone = $AC+$mAC+$AG+mAG+$AT+$mAT+$CG+$mCG+$CT+$mCT+$GT+$mGT;
#print "sumone $sumone\n";
}
elsif (($AC+$mAC+$AG+mAG+$AT+$mAT+$CG+$mCG+$CT+$mCT+$GT+$mGT) == 0) {
       $tag = "no";
       #print "$list[$i-1] NO Remainder \n";
print "$list[$i-1]# $length $GG $CC $TT $AA $GCATG $GTACG $GACTG $GTCAG $GATCG $GCTAG $GCAG
$ACTA $GTAG $GTCG $GACG $ATCA $GCTG $GATG TAG $tag \n";
#$sumtwo = $AC+$mAC+$AG+mAG+$AT+$mAT+$CG+$mCG+$CT+$mCT+$GT+$mGT;
#print "sumtwo $sumtwo\n";
}

elsif (($AC+$mAC+$AG+$mAG+$AT+$mAT+$CG+$mCG+$CT+$mCT+$GT+$mGT) > 0) {
       $tag = "yes";
#$sumthree = $AC+$mAC+$AG+mAG+$AT+$mAT+$CG+$mCG+$CT+$mCT+$GT+$mGT;
#print "sumthree $sumthree\n";

print "$list[$i-1]# $length $GG $CC $TT $AA $GCATG $GTACG $GACTG $GTCAG $GATCG $GCTAG $GCAG
$ACTA $GTAG $GTCG $GACG $ATCA $GCTG $GATG TAG $tag ";

       #print "$list[$i-1]# Remainder exists ";
       if ($AC>0) {print "AC $AC ";}
       if ($mAC>0) {print "CA $mAC ";}
       if ($AG>0) {print "AG $AG ";}
       if ($mAG>0) {print "GA $mAG ";}
       if ($AT>0) {print "AT $AT ";}
       if ($mAT>0) {print "TA $mAT ";}
       if ($CG>0) {print "CG $CG ";}
       if ($mCG >0) {print "GC $mCG ";}
       if ($CT>0) {print "CT $CT ";}
       if ($mCT >0) {print "TC $mCT ";}
       if ($GT>0) {print "GT $GT ";}
       if ($mGT >0) {print "TG $mGT ";}

       print "\n";

#print "$list[$i-1]# afterOne AC$AC CA$mAC AG$AG GA$mAG AT$AT TA$mAT
#CG$CG GC$mCG CT$CT TC$mCT GT$GT TG$mGT \n";
}
```

```
$GG=0; $CC=0; $TT=0; $AA=0; $length=0; $GCATG=0; $GTACG=0; $GACTG=0; $GTCAG=0; $GATCG=0;
$GCTAG=0; $GCAG=0; $ACTA=0; $GTAG=0; $GTCG=0; $GACG=0; $ATCA=0; $GCTG=0; $GATG=0;
$AC=0; $mAC=0; $AG=0; $mAG=0; $AT=0; $mAT=0;
$CG=0; $mCG=0; $CT=0; $mCT=0; $GT=0; $mGT=0;
#print "after summary reinitialize to zero
}
```

v. Perl program. find_max_mirror_repeats

```
#!/usr/bin/perl

# reads in a file containing a single line of text
# ESSENTIAL, line must not contain spaces
# ESSENTIAL, line must not contain end of line characters
# program outputs all mIMRs with
# greater than or equal to 50% symmetry
# NOTE. output file will not overwrite previous output file

#read in a text stream (no EOL, spaces)
print "read this file: \n";
$filename = <STDIN>;
chomp $filename;
open(DATAFILE, "$filename");
$test = <DATAFILE>;
#print "test: $test \n";
$len = length($test);
#print "len $len \n";
open (OUTFILE, ">>$filename.mIMRs");

$limit = .4999;

&get_substrings;

sub get_substrings {

   $half = int($len/2);
   $upper_limit = $len -2;

   for ($n=0; $n < $upper_limit; $n++) {
      for ($e = $len; $e > 2; $e--) {
         $a = substr($test, $n, $e);
         $la = length($a); # print "length la:$la \n";
         if ($la == $e) {
            #print "n $n e $e a $a \n";
            $temp = $a;
            &evaluate_symmetry;
         }
      }
   }
}

sub evaluate_symmetry {
   @temp_array = split(//, $temp);
   $length_temp = @temp_array;
   $half_temp = int($length_temp/2);
   $minimum_length = 6;

   #test ends
   $right_end = $length_temp - 1;;
   #print "right_end $right_end \n";
   $match=0; $nomatch=0;

   if (($temp_array[0] eq $temp_array[$right_end]) and ($right_end > $minimum_length)) {
      $end_str = $n + $e ;
      for ($left=0; $left < $half_temp; $left++) {
         $right = $length_temp - $left -1;
         if ($temp_array[$left] eq $temp_array[$right]) {
            $match++;
         }
         if ($temp_array[$left] ne $temp_array[$right]) {
```

```
            $nomatch++;
         }}
         $symmetry = $match/($match+$nomatch);
         if ($symmetry > $limit) {
               $end_str_adj = $end_str - 1;
          print "start $n end $end_str_adj temp_array: $temp length: $length_temp symmetry
$symmetry \n";
         print OUTFILE "\nstart $n end $end_str_adj  length: $length_temp symmetry
$symmetry temp_array: $temp";
         }
         # print "symmetry $symmetry \n";
         # print "\nmatch $match nomatch $nomatch symmetry $symmetry \n";
         # print "left $left $temp_array[$left] right $right $temp_array[$right] \n\n";

   }
}
close (OUTFILE);
```

vi. Perl program. find_rd_mirror_repeats

```
#!/usr/bin/perl

=pod

This program will read in a text file, determine the
location of all valid mirror repeats in the string
(>49% symmetry), and the length of the repeats.
The input file must be lower-case text in a single string -
no spaces, no new-lines, or characters other than a,c,g,t.

The program will create an output file which gives the
index values for the start and stop of each valid mirror repeat.
These values are given as a single array called @valid_in_order,
and as separate arrays for start and stop called @start2 and @end2.
The separate arrays are input arrays for the Find_Nest_Level
program and must be pasted into that program.

This program will also give the sequence position of each
dinucleotide species, although only two examples are currently
included. These examples can be easily expanded to include all
dinucleotide species.

To open the output file, first open Microsoft Word, then
open the output file from inside the Word program.

=cut

##### Read in a string. Convert to an array. ################


# reads in a text file
   print "Enter a filename: ";
   $input_file = <STDIN>;
   chomp $input_file;
   print "This is \$input_file: $input_file \n";

&input_file2string;
&string2arrayV2;
&find_XY;

# converts the input file (the $_ array) to a string
sub input_file2string {
   open(input_file);
   $input_string = <input_file>;
   close(input_file);
   chomp $input_string;
   print "This is \$input_string: $input_string.\n";
   $input_string =~ tr/A-Z/a-z/;
   print "This is lower case input_string: $input_string \n";
```

```
   }
# converts a string to an array
sub string2arrayV2 {
   @array = split(//, $input_string);
   print "This is string2arrayV2 \@array: @array.\n";
   }

print "\n";

##### Determine location of each type of dinucleotide #######

sub find_XY {
   $length = @array;
   $aa = 0;
   for ($i = 0; $i < $length; $i = $i+1) {
      if (($array[$i] eq "a") and ($array[$i+1] eq "a")) {
         $aa_location[$aa] = $i;
          $aa = $aa + 1;}}
   $ac = 0;
   for ($i = 0; $i < $length; $i = $i+1) {
      if (($array[$i] eq "a") and ($array[$i+1] eq "c")) {
          $ac_location[$ac] = $i;
          $ac = $ac + 1;}}
   $ag = 0;
   for ($i = 0; $i < $length; $i = $i+1) {
      if (($array[$i] eq "a") and ($array[$i+1] eq "g")) {
          $ag_location[$ag] = $i;
          $ag = $ag + 1;}}
   $at = 0;
   for ($i = 0; $i < $length; $i = $i+1) {
      if (($array[$i] eq "a") and ($array[$i+1] eq "t")) {
          $at_location[$at] = $i;
          $at = $at + 1;}}
   $ca = 0;
   for ($i = 0; $i < $length; $i = $i+1) {
      if (($array[$i] eq "c") and ($array[$i+1] eq "a")) {
          $ca_location[$ca] = $i;
          $ca = $ca + 1;}}
   $cc = 0;
   for ($i = 0; $i < $length; $i = $i+1) {
      if (($array[$i] eq "c") and ($array[$i+1] eq "c")) {
          $cc_location[$cc] = $i;
          $cc = $cc + 1;}}
   $cg = 0;
   for ($i = 0; $i < $length; $i = $i+1) {
      if (($array[$i] eq "c") and ($array[$i+1] eq "g")) {
          $cg_location[$cg] = $i;
          $cg = $cg + 1;}}
   $ct = 0;
   for ($i = 0; $i < $length; $i = $i+1) {
      if (($array[$i] eq "c") and ($array[$i+1] eq "t")) {
          $ct_location[$ct] = $i;
          $ct = $ct + 1;}}
   $ga = 0;
   for ($i = 0; $i < $length; $i = $i+1) {
      if (($array[$i] eq "g") and ($array[$i+1] eq "a")) {
         $ga_location[$ga] = $i;
         $ga = $ga + 1;}}
   $gc = 0;
   for ($i = 0; $i < $length; $i = $i+1) {
      if (($array[$i] eq "g") and ($array[$i+1] eq "c")) {
         $gc_location[$gc] = $i;
         $gc = $gc + 1;}}
   $gg = 0;
   for ($i = 0; $i < $length; $i = $i+1) {
      if (($array[$i] eq "g") and ($array[$i+1] eq "g")) {
         $gg_location[$gg] = $i;
         $gg = $gg + 1;}}
   $gt = 0;
   for ($i = 0; $i < $length; $i = $i+1) {
      if (($array[$i] eq "g") and ($array[$i+1] eq "t")) {
```

```
         $gt_location[$gt] = $i;
         $gt = $gt + 1;}}
   $ta = 0;
   for ($i = 0; $i < $length; $i = $i+1) {
      if (($array[$i] eq "t") and ($array[$i+1] eq "a")) {
         $ta_location[$ta] = $i;
         $ta = $ta + 1;}}
   $tc = 0;
   for ($i = 0; $i < $length; $i = $i+1) {
      if (($array[$i] eq "t") and ($array[$i+1] eq "c")) {
         $tc_location[$tc] = $i;
         $tc = $tc + 1;}}
   $tg = 0;
   for ($i = 0; $i < $length; $i = $i+1) {
      if (($array[$i] eq "t") and ($array[$i+1] eq "g")) {
         $tg_location[$tg] = $i;
         $tg = $tg + 1;}}
   $tt = 0;
   for ($i = 0; $i < $length; $i = $i+1) {
      if (($array[$i] eq "t") and ($array[$i+1] eq "t")) {
         $tt_location[$tt] = $i;
         $tt = $tt + 1;}}
}

##### Locate mirror repeats having ≥ 50% symmetry ###########

############## evaluate symmetry for aa-aa strings #########

$length_aa_location = @aa_location;
&AA_evaluate_symmetry;
sub AA_evaluate_symmetry {
   print "This is \$input_string: $input_string.\n";
   print "\@aa_location: @aa_location \n";
   # determine the number of AA-AA strings
   $num_AA_strings = $length_aa_location -1;
   print "\$num_AA_strings: $num_AA_strings \n";
   # for the number of strings bounded by reverse dinucleotides
   for ($m=0; $m < $num_AA_strings ; $m = $m + 1) {
     $start = $aa_location[$m];
     # print "\$aa_location[$m]:$aa_location[$m]   ";
     $end = $aa_location[$m+1] +1;
     # print "\$aa_location[$m+1]:$aa_location[$m+1]\n";
     print
     "aa-aa string $start..$end is ";
          for ($i=$start; $i<($end+1); $i=$i+1) {
                print $array[$i];
          } print "\n";
          $start_orig = $start; $end_orig = $end;
          # print "\$start_orig:$start_orig \$end_orig:$end_orig \n";

          $times2count = int(($end - $start + 1)/2) ;
        # print "\$times2count: $times2count \n";

   $count = 0;  # for each string, begin count at 0
   for ($b = 0; $b <  $times2count; $b = $b+1) {
      #print "\$array[$start]:$array[$start] ;
        #       \$array[$end]: $array[$end]\n";
        # evaluate symmetry by comparing ends of the string
        # after each round, increment start, decrement end
        if ($array[$start] eq $array[$end] ) {
           # count as symmetrical of they match
           $count = $count + 1;
           # print "for \$b:$b,\$count = $count \n";
           }
        $start = $start + 1; #print "\$start is: $start  ";
        $end = $end - 1;       #print "\$end is: $end \n";
        }
        # print "\$count AA: $count \n";
        # if more count is at least 50%, string is valid
        if ($count >= .5*$times2count ) {
        # create an array that identifies strings with > 50% symmetry
           $valid_start = $aa_location[$m];
```

```
          $n = $m + 1;
          $valid_stop = $aa_location[$n]; $valid_stop = $valid_stop +1;
          $valid_start_stop[$vs]=$valid_start."to".$valid_stop;
          $AA_valid_strings[$aav] = $valid_start_stop[$vs];
          $vs = $vs + 1;
          print "$start_orig to $end_orig has ≥ 50% symmetry \n";
          $aav = $aav + 1;
          }
        if ($count < .5*$times2count ) {
            print "$start_orig to $end_orig is not symmetric\n";
          }
      }
}
print "\n";
###################### evaluate symmetry for cc-cc strings ##########

$length_cc_location = @cc_location;
&CC_evaluate_symmetry;
sub CC_evaluate_symmetry {
   # determine the number of CC-CC strings
   $num_CC_strings = $length_cc_location -1;
   print "This is \$input_string: $input_string.\n";
   print "\@cc_location: @cc_location \n";
   print "\$num_CC_strings: $num_CC_strings \n";
        # for the number of strings bounded by reverse dinucleotides
        for ($m=0; $m < $num_CC_strings ; $m = $m + 1) {
        $start = $cc_location[$m] ;
        $end   = $cc_location[$m+1] +1;
        print "cc-cc string $start..$end is ";
        for ($i=$start; $i<($end+1); $i=$i+1) {
               print $array[$i];
        } print "\n";
        $start_orig = $start; $end_orig = $end;
        $times2count = int(($end - $start + 1)/2) ;
        print "\$times2count: $times2count \n";
        $count = 0;  # for each string, begin count at 0
        for ($b = 0; $b <  $times2count; $b = $b+1) {
        # print "\$array[$start]:$array[$start] ; \$array[$end]: $array[$end]\n";
           # evaluate symmetry by comparing ends of the string
           # after each round, increment start, decrement end
           if ($array[$start] eq $array[$end] ) {
              # count as symmetrical of they match
                $count = $count + 1;
                # print "for \$b:$b,\$count = $count \n";
                }
        $start = $start + 1;
        $end = $end - 1;
        }
        # if more count is at least 50%, string is valid
        if ($count >= .5*$times2count ) {
           # create an array that identifies strings with > 50% symmetry
           $valid_start = $cc_location[$m];
           $n = $m + 1; $valid_stop = $cc_location[$n];
           $valid_stop = $valid_stop +1;
           $valid_start_stop[$vs] = $valid_start."to".$valid_stop;
           $CC_valid_strings[$ccv] = $valid_start_stop[$vs] ;
         $ccv = $ccv + 1;
           $vs = $vs + 1;
           print "$start_orig to $end_orig has ≥ 50% symmetry \n";
        }
       if ($count < .5*$times2count ) {
          print "$start_orig to $end_orig is NOT symmetric\n";
        }
   }
}
print "\n";
###################### evaluate symmetry for gg-gg strings ########## works

$length_gg_location = @gg_location;
&GG_evaluate_symmetry;
sub GG_evaluate_symmetry {
   # determine the number of GG-GG strings
```

```
   $num_GG_strings = $length_gg_location -1;
   print "This is \$input_string: $input_string.\n";
   print "\@gg_location: @gg_location \n";
   print "\$num_GG_strings: $num_GG_strings \n";
   # for the number of strings bounded by reverse dinucleotides
   for ($m=0; $m < $num_GG_strings ; $m = $m + 1) {
      $start = $gg_location[$m]; $end   = $gg_location[$m+1] +1;
        print "gg-gg string $start..$end is ";
        for ($i=$start; $i<($end+1); $i=$i+1) {
           print $array[$i];
        } print "\n";
        $start_orig = $start; $end_orig = $end;
        $times2count = int(($end - $start + 1)/2) ;
        $count = 0;  # for each string, begin count at 0
        for ($b = 0; $b <  $times2count; $b = $b+1) {
           # evaluate symmetry by comparing ends of the string
           # after each round, increment start, decrement end
           if ($array[$start] eq $array[$end] ) {
         # count as symmetrical of they match
           $count = $count + 1;
           # print "for \$b:$b,\$count = $count \n";
           }
   $start = $start + 1;
   $end = $end - 1;
   }
   # if more count is at least 50%, string is valid
   if ($count >= .5*$times2count ) {
      # create an array that identifies strings with > 50% symmetry
        $valid_start = $gg_location[$m];
        $n = $m + 1; $valid_stop = $gg_location[$n];
        $valid_stop = $valid_stop +1;
        $valid_start_stop[$vs] = $valid_start."to".$valid_stop;
        $GG_valid_strings[$ggv] = $valid_start_stop[$vs] ;
      $ggv = $ggv + 1;
      $vs = $vs + 1;
        print "$start_orig to $end_orig has ≥ 50% symmetry \n";
        }
   if ($count < .5*$times2count ) {
      print "$start_orig to $end_orig is NOT symmetric\n";
        }
   }
}
print "\n";
##################### evaluate symmetry for TT-TT strings ########## works

$length_tt_location = @tt_location;
&TT_evaluate_symmetry;
sub TT_evaluate_symmetry {
   # print "Before and inside TT \$start:$start \$end:$end\n";
   # determine the number of TT-TT strings
   $num_TT_strings = $length_tt_location -1;
   print "This is \$input_string: $input_string.\n";
   print "\@tt_location: @tt_location \n";
   print "\$num_TT_strings: $num_TT_strings \n";
   # for the number of strings bounded by reverse dinucleotides
   for ($m=0; $m < $num_TT_strings ; $m = $m + 1) {
      $start = $tt_location[$m];
        $end   = $tt_location[$m+1] +1;
        print "tt-tt string $start..$end is ";
        for ($i=$start; $i<($end+1); $i=$i+1) {
           print $array[$i];
        } print "\n";
        $start_orig = $start; $end_orig = $end;
        $times2count = int(($end - $start + 1)/2) ;
        $count = 0;  # for each string, begin count at 0
        for ($b = 0; $b <  $times2count; $b = $b+1) {
        # evaluate symmetry by comparing ends of the string
        # after each round, increment start, decrement end
        if ($array[$start] eq $array[$end] ) {
           # count as symmetrical of they match
           $count = $count + 1;
           }
```

```
   $start = $start + 1;
   $end = $end - 1;
   }
   # if more count is at least 50%, string is valid
   if ($count >= .5*$times2count ) {
      # create an array that identifies strings with > 50% symmetry
        $valid_start = $tt_location[$m];
      $n = $m + 1; $valid_stop = $tt_location[$n];
        $valid_stop = $valid_stop + 1;
      $valid_start_stop[$vs] = $valid_start."to".$valid_stop;
        $TT_valid_strings[$ttv] = $valid_start_stop[$vs] ;
      $ttv = $ttv + 1;
        $vs = $vs + 1;
        print "$start_orig to $end_orig has ≥ 50% symmetry \n";
   }
   if ($count < .5*$times2count ) {
      print "$start_orig to $end_orig is NOT symmetric\n";
   }
  }
}
print "\n";
############################  evaluate symmetry for ac-ca strings  ###########

#$length_ac_location = @ac_location; $length_ca_location = @ca_location;
#print "\$length_ac_location:$length_ac_location  \$length_ca_location:$length_ca_location
\n";

&AC_evaluate_symmetry;
sub AC_evaluate_symmetry {
   print "\@ac_location: @ac_location \n";
   print "\@ca_location: @ca_location \n";
   $E=0;
   foreach $ac_location (@ac_location) {
      if ($ac_location > $ca_location[-1]) {
         last; #makes you break out of the loop
       }
       $n=0;
       while ($ac_location > $ca_location[$n]) {
          $n=$n + 1;
       }
       $ac_start[$s] = $ac_location;
       $ca_end[$E] = $ca_location[$n];
       $E=$E+1;
       $n=0;
       $num_ac_strings = $num_ac_strings + 1;
   }
   print "This is \$input_string: $input_string \n";
   # for the number of strings bounded by reverse dinucleotides
   $E=0;
   for ($m=0; $m < $num_ac_strings ; $m = $m + 1) {
        $start = $ac_location[$m]; # print "\$ac_location[$m]:$ac_location[$m]  ";
        $end   = $ca_end[$E] +1; # print "\$ca_end[$E]:$ca_end[$E]\n";
        print "ac-ca string is $start..$end ";
        for ($i=$start; $i<($end+1); $i=$i+1) {
           print $array[$i];
        } print "\n";
      $times2count = int(($end - $start + 1)/2) ;
      $count = 0;  # for each string, begin count at 0
      for ($b = 0; $b <  $times2count; $b = $b+1) {
         # evaluate symmetry by comparing ends of the string
           # after each round, increment start, decrement end
           if ($array[$start] eq $array[$end] ) {
            # count as symmetrical of they match
              $count = $count + 1;
           }
        $start = $start + 1;
        $end = $end - 1;
   }
   # if more count is at least 50%, string is valid
   if ($count >= .5*$times2count ) {
      # create an array that identifies strings with ≥ 50% symmetry
        $valid_start = $ac_location[$m]; $valid_stop = $ca_end[$E];
```

```
        $valid_stop = $valid_stop + 1;
        $valid_start_stop[$vs] = $valid_start."to".$valid_stop;
        $AC_valid_strings[$acv] = $valid_start_stop[$vs] ;
      $acv = $acv + 1;
        $vs = $vs + 1;
        $ca_end[$E] = $ca_end[$E] + 1;
        print "$ac_location[$m] to $ca_end[$E] has ≥ 50% symmetry";
        $ca_end[$E] = $ca_end[$E] - 1;
   }
   if ($count < .5*$times2count ) {
        $ca_end[$e] = $ca_end[$e] + 1;
        print "The string $ac_location[$m] to $ca_end[$E] is NOT symmetric\n";
        $ca_end[$E] = $ca_end[$E] - 1;
   }
   $E= $E+1;
   print "\n";
   }
}
print "\n";
############################  evaluate symmetry for ag-ga strings  ###########3

&AG_evaluate_symmetry;
sub AG_evaluate_symmetry {
   print "\@ag_location: @ag_location \n";
   print "\@ga_location: @ga_location \n";
   $E=0;
   foreach $ag_location (@ag_location) {
      if ($ag_location > $ga_location[-1]) {
          last; #makes you break out of the loop
        }
        $n=0;
        while ($ag_location > $ga_location[$n]) {
           $n=$n + 1;
        }
        $ag_start[$s] = $ag_location; #print "\$ag_location:$ag_location\n";
        $ga_end[$E] = $ga_location[$n];
      #print "\$E:$E  \$ga_location[$n]:$ga_location[$n] \n";
        $E=$E+1;
        $n=0;  #print "\$n: $n \n";
        $num_ag_strings = $num_ag_strings + 1;
   }
   print "\$num_ag_strings: $num_ag_strings \n";
   # for the number of strings bounded by reverse dinucleotides
   $E=0;
   for ($m=0; $m < $num_ag_strings ; $m = $m + 1) {
        $start = $ag_location[$m]; # print "\$ag_location[$m]:$ag_location[$m] ";
        $end   = $ga_end[$E] +1; # print "\$ga_end[$E]:$ga_end[$E]\n";
        print "ag-ga string is $start..$end ";
        for ($i=$start; $i<($end+1); $i=$i+1) {
         print $array[$i];
        } print "\n";
        $times2count = int(($end - $start + 1)/2) ;
        $count = 0;  # for each string, begin count at 0
        for ($b = 0; $b <  $times2count; $b = $b+1) {
           # evaluate symmetry by comparing ends of the string
           # after each round, increment start, decrement end
           if ($array[$start] eq $array[$end] ) { #6s
              # count as symmetrical of they match
                $count = $count + 1;
           }
        $start = $start + 1;
        $end = $end - 1;
        }
        # if more count is at least 50%, string is valid
        if ($count >= .5*$times2count ) {
           # create an array that identifies strings with > 50% symmetry
           $valid_start = $ag_location[$m]; $valid_stop = $ga_end[$E];
           $valid_stop = $valid_stop + 1;
           $valid_start_stop[$vs] = $valid_start."to".$valid_stop;
           $AG_valid_strings[$agv] = $valid_start_stop[$vs] ;
         $agv = $agv + 1;
           $vs = $vs + 1;
```

```
            $ga_end[$E] = $ga_end[$E] + 1;
            print "$ag_location[$m] to $ga_end[$E] has ≥ 50% symmetry";
                $ga_end[$E] = $ga_end[$E] - 1;
            }
          if ($count < .5*$times2count ) {
           $ga_end[$e] = $ga_end[$e] + 1;
            print "The string $ag_location[$m] to $ga_end[$E] is NOT symmetric\n";
            $ga_end[$E] = $ga_end[$E] - 1;
          }
          $E= $E+1;
          print "\n";
    }
}
print "\n";
############################  evaluate symmetry for at-ta strings  ###########3

&AT_evaluate_symmetry;
sub AT_evaluate_symmetry {
    print "\@at_location: @at_location \n";
    print "\@ta_location: @ta_location \n";
    $E=0;
    foreach $at_location (@at_location) {
       if ($at_location > $ta_location[-1]) {
            last; #makes you break out of the loop
         }
         $n=0;
         while ($at_location > $ta_location[$n]) {
            $n=$n + 1;
         }
         $at_start[$s] = $at_location; #print "\$at_location:$at_location\n";
         $ta_end[$E] = $ta_location[$n];
       #print "\$E:$E  \$ta_location[$n]:$ta_location[$n] \n";
         $E=$E+1;
         $n=0;  #print "\$n: $n \n";
         $num_at_strings = $num_at_strings + 1;
    }
    print "\$num_at_strings: $num_at_strings \n";
    print "This is \$input_string:$input_string\n";
    # for the number of strings bounded by reverse dinucleotides
    $E=0;
    for ($m=0; $m < $num_at_strings ; $m = $m + 1) {
       $start = $at_location[$m];
         $end   = $ta_end[$E] +1;
         print "at-ta string $start..$end ";
         for ($i=$start; $i<($end+1); $i=$i+1) {
                print $array[$i];
         } print "\n";
         $times2count = int(($end - $start + 1)/2) ;
         $count = 0;  # for each string, begin count at 0
         for ($b = 0; $b <  $times2count; $b = $b+1) {
            # evaluate symmetry by comparing ends of the string
            # after each round, increment start, decrement end
            if ($array[$start] eq $array[$end] ) {
               # count as symmetrical of they match
                 $count = $count + 1;
            }
            $start = $start + 1;
            $end = $end - 1;
         }
         # if more count is at least 50%, string is valid
         if ($count >= .5*$times2count ) {
                # create an array that identifies strings with > 50% symmetry
                $valid_start = $at_location[$m]; $valid_stop = $ta_end[$E];
                $valid_stop = $valid_stop + 1;
                $valid_start_stop[$vs] = $valid_start."to".$valid_stop;
              $AT_valid_strings[$atv] = $valid_start_stop[$vs] ;
             $atv = $atv + 1;
                $vs = $vs + 1;
                $ta_end[$E] = $ta_end[$E] + 1;
                print "$at_location[$m] to $ta_end[$E] has≥ 50% symmetry";
                $ta_end[$E] = $ta_end[$E] - 1;
         }
```

```
        if ($count < .5*$times2count ) {
                $ta_end[$e] = $ta_end[$e] + 1;
                print "$at_location[$m] to $ta_end[$E] is NOT symmetric";
                $ta_end[$E] = $ta_end[$E] - 1;
        }
        $E= $E+1;
        print "\n";
    }
}
print "\n";
############################  evaluate symmetry for ca-ac strings  ###########3

&CA_evaluate_symmetry;
sub CA_evaluate_symmetry {
    print "\@ca_location: @ca_location \n";
    print "\@ac_location: @ac_location \n";
    $E=0;
    foreach $ca_location (@ca_location) {
       if ($ca_location > $ac_location[-1]) {
          last;
        }
       $n=0;
       while ($ca_location > $ac_location[$n]) {
       $n=$n + 1;
       }
    $ca_start[$s] = $ca_location; #print "\$ca_location:$ca_location\n";
    $ac_end[$E] = $ac_location[$n];
    #print "\$E:$E  \$ac_location[$n]:$ac_location[$n] \n";
    $E=$E+1;
    $n=0;
    $num_ca_strings = $num_ca_strings + 1;
    }
    print "\$num_ca_strings: $num_ca_strings  \n";
    # for the number of strings bounded by reverse dinucleotides
    $E=0;
    for ($m=0; $m < $num_ca_strings ; $m = $m + 1) {
         $start = $ca_location[$m]; # print "\$ca_location[$m]:$ca_location[$m]  ";
         $end   = $ac_end[$E] +1; # print "\$ac_end[$E]:$ac_end[$E]\n";
         print "ca-ac string $start..$end is ";
         for ($i=$start; $i<($end+1); $i=$i+1) {
                print $array[$i];
         } print "\n";
         $times2count = int(($end - $start + 1)/2) ;
         $count = 0;  # for each string, begin count at 0
         for ($b = 0; $b <  $times2count; $b = $b+1) {
            # evaluate symmetry by comparing ends of the string
            # after each round, increment start, decrement end
            if ($array[$start] eq $array[$end] ) {
               # count as symmetrical of they match
               $count = $count + 1;
                 }
            $start = $start + 1;
            $end = $end - 1;
         }
         # if more count is at least 50%, string is valid
         if ($count >= .5*$times2count ) {
            # create an array that identifies strings with > 50% symmetry
            $valid_start = $ca_location[$m]; $valid_stop = $ac_end[$E];
            $valid_stop = $valid_stop + 1;
            $valid_start_stop[$vs] = $valid_start."to".$valid_stop;
            $CA_valid_strings[$cav] = $valid_start_stop[$vs] ;
          $cav = $cav + 1;
            $vs = $vs + 1;
            $ac_end[$E] = $ac_end[$E] + 1;
            print "$ca_location[$m] to $ac_end[$E] has ≥ 50% symmetry";
            $ac_end[$E] = $ac_end[$E] - 1;
         }
         if ($count < .5*$times2count ) {
            $ac_end[$e] = $ac_end[$e] + 1;
            print "The string $ca_location[$m] to $ac_end[$E] is NOT symmetric";
            $ac_end[$E] = $ac_end[$E] - 1;
            }
```

```
        $E= $E+1;
        print "\n";
    }
}
print "\n";
############################  evaluate symmetry for cg-gc strings  ###########3

&CG_evaluate_symmetry;
sub CG_evaluate_symmetry {
    print "\@cg_location: @cg_location \n";
    print "\@gc_location: @gc_location \n";
    $E=0;
    foreach $cg_location (@cg_location) {
       if ($cg_location > $gc_location[-1]) {
         last;
         }
         $n=0;
         while ($cg_location > $gc_location[$n]) {
            $n=$n + 1;
         }
         $cg_start[$s] = $cg_location; #print "\$cg_location:$cg_location\n";
         $gc_end[$E] = $gc_location[$n];
       #print "\$E:$E  \$gc_location[$n]:$gc_location[$n] \n";
         $E=$E+1;
         $n=0;  #print "\$n: $n \n";
         $num_cg_strings = $num_cg_strings + 1;
    }
    print "\$num_cg_strings: $num_cg_strings \n";
    print "This is \$input_string: $input_string.\n";
    # for the number of strings bounded by reverse dinucleotides
    $E=0;
    for ($m=0; $m < $num_cg_strings ; $m = $m + 1) {
       $start = $cg_location[$m]; # print "\$cg_location[$m]:$cg_location[$m]  ";
         $end   = $gc_end[$E] +1; #print "\$gc_end[$E]:$gc_end[$E]\n";
         # print "cg-gc string is $start..$end ";
         for ($i=$start; $i<($end+1); $i=$i+1) {
            print $array[$i];
         } print "\n";
         $times2count = int(($end - $start + 1)/2) ;
         $count = 0;  # for each string, begin count at 0
         for ($b = 0; $b <  $times2count; $b = $b+1) {
            # evaluate symmetry by comparing ends of the string
            # after each round, increment start, decrement end
            if ($array[$start] eq $array[$end] ) {
               # count as symmetrical of they match
                 $count = $count + 1;
            }
            $start = $start + 1;
            $end = $end - 1;
         }
         # if more count is at least 50%, string is valid
         if ($count >= .5*$times2count ) {
                # create an array that identifies strings with > 50% symmetry
                $valid_start = $cg_location[$m]; $valid_stop = $gc_end[$E];
                $valid_stop = $valid_stop + 1;
                $valid_start_stop[$vs] = $valid_start."to".$valid_stop;
              $CG_valid_strings[$cgv] = $valid_start_stop[$vs] ;
            $cgv = $cgv + 1;
                $vs = $vs + 1;
                $gc_end[$E] = $gc_end[$E] + 1;
                print "$cg_location[$m] to $gc_end[$E] has ≥ 50% symmetry \n";
                $gc_end[$E] = $gc_end[$E] - 1;
                }
         if ($count < .5*$times2count ) {
                $gc_end[$e] = $gc_end[$e] + 1;
                print "$cg_location[$m] to $gc_end[$E] is NOT symmetric\n";
                $gc_end[$E] = $gc_end[$E] - 1;
                }
         $E= $E+1;
         }
}
print "\n";
```

```
############################  evaluate symmetry for ct-tc strings  ###########3

&CT_evaluate_symmetry;
sub CT_evaluate_symmetry {
   print "\@ct_location: @ct_location \n";
   print "\@tc_location: @tc_location \n";
   $E=0;
   foreach $ct_location (@ct_location) {
      if ($ct_location > $tc_location[-1]) {
          last; #makes you break out of the loop
       }
       $n=0;
       while ($ct_location > $tc_location[$n]) {
          $n=$n + 1;
       }
       $ct_start[$s] = $ct_location; #print "\$ct_location:$ct_location\n";
       $tc_end[$E] = $tc_location[$n];
       #print "\$E:$E  \$tc_location[$n]:$tc_location[$n] \n";
       $E=$E+1;
       $n=0;  #print "\$n: $n \n";
       $num_ct_strings = $num_ct_strings + 1;
       }
   print "\$num_ct_strings: $num_ct_strings \n";
   print "This is \$input_string: $input_string.\n";
   # for the number of strings bounded by reverse dinucleotides
   $E=0;
   for ($m=0; $m < $num_ct_strings ; $m = $m + 1) {
        $start = $ct_location[$m]; # print "\$ct_location[$m]:$ct_location[$m]  ";
        $end   = $tc_end[$E] +1; # print "\$tc_end[$E]:$tc_end[$E]\n";
        print "ct-tc string $start..$end ";
        for ($i=$start; $i<($end+1); $i=$i+1) {
              print $array[$i];
        }  print "\n";
        $times2count = int(($end - $start + 1)/2) ;
        $count = 0;  # for each string, begin count at 0
        for ($b = 0; $b <  $times2count; $b = $b+1) {
           # evaluate symmetry by comparing ends of the string
           # after each round, increment start, decrement end
           if ($array[$start] eq $array[$end] ) {
              # count as symmetrical of they match
                $count = $count + 1;
           }
        $start = $start + 1;
        $end = $end - 1;
        }
        # if more count is at least 50%, string is valid
        if ($count >= .5*$times2count ) {
           # create an array that identifies strings with > 50% symmetry
           $valid_start = $ct_location[$m]; $valid_stop = $tc_end[$E];
           $valid_stop = $valid_stop + 1;
           $valid_start_stop[$vs] = $valid_start."to".$valid_stop;
           $CT_valid_strings[$ctv] = $valid_start_stop[$vs] ;
         $ctv = $ctv + 1;
           $vs = $vs + 1;
           $tc_end[$E] = $tc_end[$E] + 1;
           print "$ct_location[$m] to $tc_end[$E] has ≥ 50% symmetry \n";
                $tc_end[$E] = $tc_end[$E] - 1;
           }
        if ($count < .5*$times2count ) {
         $tc_end[$e] = $tc_end[$e] + 1;
           print "$ct_location[$m] to $tc_end[$E] is NOT symmetric\n";
                $tc_end[$E] = $tc_end[$E] - 1;
                }
        $E= $E+1;
   }
}
print "\n";
############################  evaluate symmetry for ga-ag strings  ###########3

&GA_evaluate_symmetry;
sub GA_evaluate_symmetry {
   print "\@ga_location: @ga_location \n";
```

```
   print "\@ag_location: @ag_location \n";
   $E=0;
   foreach $ga_location (@ga_location) {
      if ($ga_location > $ag_location[-1]) {
          last;
      }
      $n=0;
        while ($ga_location > $ag_location[$n]) {
           $n=$n + 1;
        }
        $ga_start[$s] = $ga_location; #print "\$ga_location:$ga_location\n";
        $ag_end[$E] = $ag_location[$n];
      #print "\$E:$E  \$ag_location[$n]:$ag_location[$n] \n";
        $E=$E+1;
        $n=0;  #print "\$n: $n \n";
        $num_ga_strings = $num_ga_strings + 1;
        }
      print "\$num_ga_strings: $num_ga_strings \n";
      print "This is \$input_string: $input_string \n";
      # for the number of strings bounded by reverse dinucleotides
      $E=0;
      for ($m=0; $m < $num_ga_strings ; $m = $m + 1) {
          $start = $ga_location[$m];
         $end   = $ag_end[$E] +1;
         print "ga-ag string $start..$end is ";
          for ($i=$start; $i<($end+1); $i=$i+1) {
              print $array[$i];
          } print "\n";
        $times2count = int(($end - $start + 1)/2) ;
        $count = 0;  # for each string, begin count at 0
        for ($b = 0; $b <  $times2count; $b = $b+1) {
           if ($array[$start] eq $array[$end] ) {
              # count as symmetrical of they match
                $count = $count + 1;
           }
        $start = $start + 1;
        $end = $end - 1;
        }
        # if more count is at least 50%, string is valid
        if ($count >= .5*$times2count ) {
           # create an array that identifies strings with > 50% symmetry
           $valid_start = $ga_location[$m]; $valid_stop = $ag_end[$E];
           $valid_stop = $valid_stop + 1;
           $valid_start_stop[$vs] = $valid_start."to".$valid_stop;
           $GA_valid_strings[$gav] = $valid_start_stop[$vs] ;
         $gav = $gav + 1;
           $vs = $vs + 1;
           $ag_end[$E] = $ag_end[$E] + 1;
           print "$ga_location[$m] to $ag_end[$E] has ≥ 50% symmetry \n";
               $ag_end[$E] = $ag_end[$E] - 1;
           }
        if ($count < .5*$times2count ) {
           $ag_end[$e] = $ag_end[$e] + 1;
           print "The string $ga_location[$m] to $ag_end[$E] is NOT symmetric\n";
           $ag_end[$E] = $ag_end[$E] - 1;
        }
   $E= $E+1;
   }
}
print "\n";
##########################  evaluate symmetry for gc-cg strings  ###########3

#\$length_cg_location:$length_cg_location \n";
&GC_evaluate_symmetry;
sub GC_evaluate_symmetry {
   print "\@gc_location: @gc_location \n";
   print "\@cg_location: @cg_location \n";
   $E=0;
   foreach $gc_location (@gc_location) {
      if ($gc_location > $cg_location[-1]) {
          last;
       }
```

```
        $n=0;
        while ($gc_location > $cg_location[$n]) {
           $n=$n + 1;
        }
        $gc_start[$s] = $gc_location;
        $cg_end[$E] = $cg_location[$n];
      $E=$E+1;
        $n=0;
        $num_gc_strings = $num_gc_strings + 1;
   }
   print "\$num_gc_strings: $num_gc_strings \n";
   print "This is \$input_string: $input_string \n";
   $E=0;
   for ($m=0; $m < $num_gc_strings ; $m = $m + 1) {
        $start = $gc_location[$m];
        $end   = $cg_end[$E] +1;
        print "gc-cg string $start..$end is ";
        for ($i=$start; $i<($end+1); $i=$i+1) {
           print $array[$i];
        } print "\n";
        $times2count = int(($end - $start + 1)/2) ;
        $count = 0;
        for ($b = 0; $b < $times2count; $b = $b+1) {
           if ($array[$start] eq $array[$end] ) {
              # count as symmetrical of they match
                $count = $count + 1;
           }
        $start = $start + 1;
        $end = $end - 1;
        }
        # if more count is at least 50%, string is valid
        if ($count >= .5*$times2count ) {
           # create an array that identifies strings with > 50% symmetry
           $valid_start = $gc_location[$m]; $valid_stop = $cg_end[$E];
           $valid_stop = $valid_stop + 1;
           $valid_start_stop[$vs] = $valid_start."to".$valid_stop;
           $GC_valid_strings[$gcv] = $valid_start_stop[$vs] ;
         $gcv = $gcv + 1;
           $vs = $vs + 1;
           $cg_end[$E] = $cg_end[$E] + 1;
           print "$gc_location[$m] to $cg_end[$E] has ≥ 50% symmetry \n";
                $cg_end[$E] = $cg_end[$E] - 1;
           }
        if ($count < .5*$times2count ) {
           $cg_end[$e] = $cg_end[$e] + 1;
           print "$gc_location[$m] to $cg_end[$E] is NOT symmetric\n";
           $cg_end[$E] = $cg_end[$E] - 1;
        }
        $E= $E+1;
   }
}
print "\n";

############################  evaluate symmetry for gt-tg strings  ###########3

&GT_evaluate_symmetry;
sub GT_evaluate_symmetry {
   print "\@gt_location: @gt_location \n";
   print "\@tg_location: @tg_location \n";
   $E=0;
   foreach $gt_location (@gt_location) {
      if ($gt_location > $tg_location[-1]) {
           last;
        }
        $n=0;
        while ($gt_location > $tg_location[$n]) {
           $n=$n + 1;
        }
        $gt_start[$s] = $gt_location;
        $tg_end[$E] = $tg_location[$n];
      $E=$E+1;
        $n=0;
```

```
        $num_gt_strings = $num_gt_strings + 1;
   }
   print "\$num_gt_strings: $num_gt_strings \n";
   print "This is \$input_string: $input_string \n";
   $E=0;
   for ($m=0; $m < $num_gc_strings ; $m = $m + 1) {
        $start = $gt_location[$m];
        $end   = $tg_end[$E] +1;
        print "gt-tg string $start..$end is ";
        for ($i=$start; $i<($end+1); $i=$i+1) {
           print $array[$i];
        } print "\n";
        $times2count = int(($end - $start + 1)/2) ;
        $count = 0;
        for ($b = 0; $b < $times2count; $b = $b+1) {
           if ($array[$start] eq $array[$end] ) {
              # count as symmetrical of they match
                $count = $count + 1;
           }
        $start = $start + 1;
        $end = $end - 1;
        }
        # if more count is at least 50%, string is valid
        if ($count >= .5*$times2count ) {
           # create an array that identifies strings with > 50% symmetry
           $valid_start = $gt_location[$m]; $valid_stop = $tg_end[$E];
           $valid_stop = $valid_stop + 1;
           $valid_start_stop[$vs] = $valid_start."to".$valid_stop;
           $GT_valid_strings[$gtv] = $valid_start_stop[$vs] ;
         $gtv = $gtv + 1;
           $vs = $vs + 1;
           $tg_end[$E] = $tg_end[$E] + 1;
           print "$gt_location[$m] to $tg_end[$E] has ≥ 50% symmetry \n";
                $tg_end[$E] = $tg_end[$E] - 1;
           }
        if ($count < .5*$times2count ) {
           $tg_end[$e] = $tg_end[$e] + 1;
           print "$gt_location[$m] to $tg_end[$E] is NOT symmetric\n";
           $tg_end[$E] = $tg_end[$E] - 1;
        }
        $E= $E+1;
   }
}
print "\n";

############################  evaluate symmetry for ta-at strings  ###########3

&TA_evaluate_symmetry;
sub TA_evaluate_symmetry {
   print "\@ta_location: @ta_location \n";
   print "\@at_location: @at_location \n";
   $E=0;
   foreach $ta_location (@ta_location) {
      if ($ta_location > $at_location[-1]) {
           last;
        }
        $n=0;
        while ($ta_location > $at_location[$n]) {
           $n=$n + 1;
        }
        $ta_start[$s] = $ta_location;
        $at_end[$E] = $at_location[$n];
      $E=$E+1;
        $n=0;
        $num_ta_strings = $num_ta_strings + 1;
        }
      print "\$num_ta_strings: $num_ta_strings \n";
      # for the number of strings bounded by reverse dinucleotides
      $E=0;
      for ($m=0; $m < $num_ta_strings ; $m = $m + 1) {
           $start = $ta_location[$m];
        $end   = $at_end[$E] +1;
```

```
            print "ta-at string $start..$end is ";
            for ($i=$start; $i<($end+1); $i=$i+1) {
                print $array[$i];
            }
         $times2count = int(($end - $start + 1)/2) ;
         $count = 0;  # for each string, begin count at 0
         for ($b = 0; $b <  $times2count; $b = $b+1) {
            if ($array[$start] eq $array[$end] ) {
               $count = $count + 1;
            }
            $start = $start + 1;
            $end = $end - 1;
         }
         if ($count >= .5*$times2count ) {
            $valid_start = $ta_location[$m]; $valid_stop = $at_end[$E];
            $valid_stop = $valid_stop + 1;
            $valid_start_stop[$vs] = $valid_start."to".$valid_stop;
            $TA_valid_strings[$tav] = $valid_start_stop[$vs] ;
          $tav = $tav + 1;
            $vs = $vs + 1;
            $at_end[$E] = $at_end[$E] + 1;
            print "$ta_location[$m] to $at_end[$E] has ≥ 50% symmetry \n";
            $at_end[$E] = $at_end[$E] - 1;
         }
         if ($count < .5*$times2count ) {
            $at_end[$e] = $at_end[$e] + 1;
            print "$ta_location[$m] to $at_end[$E] is NOT symmetric\n";
            $at_end[$E] = $at_end[$E] - 1;
         }
         $E= $E+1;
   }
}
print "\n";
##########################  evaluate symmetry for tc-ct strings  ##########3

&TC_evaluate_symmetry;
sub TC_evaluate_symmetry {
   print "\@tc_location: @tc_location \n";
   print "\@ct_location: @ct_location \n";
   $E=0;
   foreach $tc_location (@tc_location) {
      if ($tc_location > $ct_location[-1]) {
          last;
       }
       $n=0;
       while ($tc_location > $ct_location[$n]) {
          $n=$n + 1;
       }
       $tc_start[$s] = $tc_location;
       $ct_end[$E] = $ct_location[$n];
       $E=$E+1;
       $n=0;  #print "\$n: $n \n";
       $num_tc_strings = $num_tc_strings + 1;
   }
   print "\$num_tc_strings: $num_tc_strings \n";
   $E=0;
   for ($m=0; $m < $num_tc_strings ; $m = $m + 1) {
        $start = $tc_location[$m];
        $end   = $ct_end[$E] +1;
        print "tc-ct string $start..$end is ";
        for ($i=$start; $i<($end+1); $i=$i+1) {
           print $array[$i];
      }
        $times2count = int(($end - $start + 1)/2) ;
        $count = 0;
        for ($b = 0; $b <  $times2count; $b = $b+1) {
         if ($array[$start] eq $array[$end] ) {
              $count = $count + 1;
                }
           $start = $start + 1;
           $end = $end - 1;
        }
```

```
        if ($count >= .5*$times2count ) {
           $valid_start = $tc_location[$m]; $valid_stop = $ct_end[$E];
           $valid_stop = $valid_stop + 1;
           $valid_start_stop[$vs] = $valid_start."to".$valid_stop;
           $TC_valid_strings[$tcv] = $valid_start_stop[$vs] ;
         $tcv = $tcv + 1;
           $vs = $vs + 1;
           $ct_end[$E] = $ct_end[$E] + 1;
           print "$tc_location[$m] to $ct_end[$E] has ≥ 50% symmetry \n";
           $ct_end[$E] = $ct_end[$E] - 1;
        }
        if ($count < .5*$times2count ) {
           $ct_end[$e] = $ct_end[$e] + 1;
           print "$tc_location[$m] to $ct_end[$E] is NOT symmetric\n";
               $ct_end[$E] = $ct_end[$E] - 1;
        }
        $E= $E+1;
   }
}
print "\n";
############################  evaluate symmetry for tg-gt strings  ###########

&TG_evaluate_symmetry;
sub TG_evaluate_symmetry {
   print "\@tg_location: @tg_location \n";
   print "\@gt_location: @gt_location \n";
   $E=0;
   foreach $tg_location (@tg_location) {
      if ($tg_location > $gt_location[-1]) {
          last;
        }
        $n=0;
        while ($tg_location > $gt_location[$n]) {
           $n=$n + 1;
        }
        $tg_start[$s] = $tg_location;
        $gt_end[$E] = $gt_location[$n];
        $E=$E+1;
        $n=0;
        $num_tg_strings = $num_tg_strings + 1;
   }
   print "\$num_tg_strings: $num_tg_strings \n";
   $E=0;
   for ($m=0; $m < $num_tg_strings ; $m = $m + 1) {
        $start = $tg_location[$m];
        $end   = $gt_end[$E] +1;
        print "tg-gt string $start..$end is \n";
        for ($i=$start; $i<($end+1); $i=$i+1) {
           print $array[$i];
           }
        $times2count = int(($end - $start + 1)/2) ;
        $count = 0;
        for ($b = 0; $b <  $times2count; $b = $b+1) {
          if ($array[$start] eq $array[$end] ) {
          $count = $count + 1;
           }
      $start = $start + 1;
        $end = $end - 1;
      }
      if ($count >= .5*$times2count ) {
           $valid_start = $tg_location[$m]; $valid_stop = $gt_end[$E];
           $valid_stop = $valid_stop + 1;
           $valid_start_stop[$vs] = $valid_start."to".$valid_stop;
           $TG_valid_strings[$tgv] = $valid_start_stop[$vs] ;
         $tgv = $tgv + 1;
           $vs = $vs + 1;
           $gt_end[$E] = $gt_end[$E] + 1;
           print "$tg_location[$m] to $gt_end[$E] has ≥ 50% symmetry \n";
           $gt_end[$E] = $gt_end[$E] - 1;
        }
        if ($count < .5*$times2count ) {
           $gt_end[$e] = $gt_end[$e] + 1;
```

```
            print "$tg_location[$m] to $gt_end[$E] is NOT symmetric\n";
            $gt_end[$E] = $gt_end[$E] - 1;
         }
         $E= $E+1;
   }
}
print "\n";

### SORTS AN ARRAY - CONVERTS @valid_start_stop to @valid_in_order #######

&sort_by_index;
sub sort_by_index {
   @valid_in_order = sort by_number @valid_start_stop;
   sub by_number {
      if ($a < $b) {
      return -1;
      }
      if ($a == $b) {
      return 0;
      }
      if ($a > $b) {
      return 1;
     }
   }
   foreach ($valid_in_order) {
      $valid_in_order[$vio] = "\t".$valid_in_order[$vio];
      #print OUT "$valid_in_order[$vio] \n ";
   }
}

### SPLITS @valid_in_order into @start2 and @end2 #########

$length_valid_in_order = @valid_in_order;
print "\$length_valid_in_order:$length_valid_in_order \n";
for ($m=0; $m < ($length_valid_in_order); $m = $m + 1) {
   ($start2[$m], $end2[$m]) = split(/to/,$valid_in_order[$m]);
   print "\$start2[$m]:$start2[$m]    \$end2[$m]:$end2[$m] \n";
}

### Write to file named $input_file.out ######################

&write_to_file;

sub write_to_file {
   open(OUT, ">$input_file.rdIMRs.out") ||die "Cannot create outfile \n";

   # print out location of each type of dinucleotide
   print OUT "\@valid_in_order: @valid_in_order \n";
   print OUT "\@start2: @start2 \n";
   print OUT "\@end2: @end2 \n";
   #print OUT "\@ag_location: @ag_location \n";
   #print OUT "\@ga_location: @ga_location \n";

   print OUT "\n";
   print OUT "\n";

close (OUT) || die "Cannot close $input_file.out \n";
}

print "\n - The End - \n";
```

vii. Perl program. nest_repeats

```
#!/usr/bin/perl

# Dorothy Lang and Chuck Calef jointly created this program.

=pod

This program evaluates the nesting levels of mirror repeats.
The array of valid mirror repeats was previously determined
by the <find_max_mirror_repeats> or <find_rd_mirror_repeats>
program, and broken into separate arrays for the start and
stop of each mirror repeat. The result is that
$start2[0]..$end2[0] is the start and stop of the first
valid mirror repeat, $start2[1]..$end2[1] is the start and stop
of the second valid mirror repeat, etc.

The @start2 and @end2 arrays must be pasted into this program,
from the output of the <find..mirror_repeats> program. The output
or the <find..mirror_repeats> program will be of the form:

@start2: 0 2 3 4
@end2: 6 13 24 8

This output must be converted to the correct format for
initializing an array:

@start2 = qw( 0 2 3 4 );
@end2 = qw( 6 13 24 8 );

Two output files will be created that can be directly imported
into an EXCEL spreadsheet. The output file <nested.nums.out>
gives the start and stop, length and nest level for each IMR,
solely as numbers. The output file <nested.words.out> gives the
start and stop, length and nest level with the variable name
written out.

=cut


############################################################
# paste new @start2 and @end2 into array lists below

@start2 = qw( 0 2 3 4);

@end2 = qw( 6 13 24 8);

open (OUTNUMS, ">nested.nums.out") || die "Can't open nested.nums.out";
open (OUTWORDS, ">nested.words.out") || die "Can't open nested.words.out";

&StoreNestLevels;

&find_subsets;

sub StoreNestLevels{
#   Go through the 2 data arrays and assign to each start and stop pair
#   a NestLevel which is stored in a new array called NestLevels

    local($PrevStart, $PrevEnd, $start, $end, $NestLevel, $i, $j);
#   read a pair of start and end values
    $PrevStart = $start2[0];  $PrevEnd = $end2[0];

    $NestLevels[0] = 0;  $NestLevel = 0;

#   $#end2 is the index value of the last element of fred
    for($i = 0; $i <= $#end2; $i++){
        $start = $start2[$i];  $end = $end2[$i];

        # if this pair is nested within previous pair
        if($start >= $PrevStart && $end <= $PrevEnd){
            $NestLevel++;
            $NestLevels[$i] = $NestLevel;
```

```
            $PrevStart = $start;
            $PrevEnd = $end;
        }

        # if this pair extends beyond the end of previous pair
        elsif($start >= $PrevStart && $end > $PrevEnd){
            # count backwards
            for($j = $i-2; $j >= 0; $j--){
                # find the nest level above the current level
                if($NestLevels[$j] < $NestLevel){
                    # if this end extends beyone end of next higher level
                    if($end > $end2[$j]){
                        # decrease the current nest level
                        $NestLevel--;
                        # go check the next higher nest level
                        next;
                    }
                last;
             }
        }
        # store the current nest level
        $NestLevels[$i] = $NestLevel;
        $PrevStart = $start;
        $PrevEnd = $end;
    }
    # organize output by nest level
    $amt = $NestLevels[$i];
    $tabit = "\t" x $amt ;
    $length = ($end2[$i] - $start2[$i] + 1 );
    print OUTWORDS "\$i:$i $tabit $start2[$i]..$end2[$i] level:$NestLevels[$i]
length:$length \n";
  $dotit = "." x $amt; #$nest_dots = $NestLevels[$i];
    print OUTNUMS
"\$i:$i$dotit$start2[$i].$end2[$i].......$NestLevels[$i].......$length\n";

}
}


&find_subsets;

sub find_subsets{
    # this sub finds nested pairs of data points
    # each "." stands for a column in an Excel spreadsheet

    # local variables:
    local($bigend, $start, $end, $nextend, $nextstart, $SpaceBetween, $BetweenPadding,
$BeforePadding,$loopcount);

    $bigend = 0;
    $loopcount = -1;
    # shift end values off the array
    OUTER: while($end = shift(@end2)){
    $loopcount++;
    $start = shift(@start2);
    # distance between start and end
    $SpaceBetween = $end - $start - 1;
    # distance represented by dots
    $BetweenPadding = "." x $SpaceBetween;
    # dots also needed in front of data pairs
    $BeforePadding =  "." x ($start - 1);

    # bigend is highest end value
    if($end > $bigend){
        $bigend = $end;
    }
    while($nextend = shift(@end2)){
        $loopcount++;
        # this finds the next unnested end
        if($nextend > $bigend){
            # put this end value back on the stack
            unshift(@end2, $nextend);
```

```
            next OUTER;
        }
        else {
            # this will be a nested value
            $nextstart = shift(@start2);
            $SpaceBetween = $nextend - $nextstart - 1;
            $BetweenPadding = "." x $SpaceBetween;
            $BeforePadding =  "." x ($nextstart - 1);
# print "$BeforePadding|$BetweenPadding|  $nextstart-$nextend
#($NestLevels[$loopcount])\n";
# print out an unnested pair of values
        }
    }
}
}
```

The four published papers cited below have been removed from the e-theses due to copyright restrictions:

Lang, DM. 2007. Circuit Assemblages Derived From Net Dinucleotide Values Provide A Succinct Identity For The HIV-1 Genome And Each Of Its Genes. *Virus Genes*. 36(1):11-26. PMID: 17987374. DOI: 10.1007/s11262-007-0128-6.

Lang, DM. 2007. Imperfect DNA mirror repeats in the gag gene of HIV-1 (HXB2) coincide with protein structural elements and functional domains in each of the cleaved mature proteins. *Virology J.* 4(1): 113. PMID: 17963512, DOI: 10.1186/1743-422X-4-113.

Lang DM. 2005. Imperfect DNA mirror repeats in E. coli TnsA and other protein-coding DNA. 2005. *Biosystems*. 81(3): 183-207, DOI: 10.1016/j.biosystems.2005.03.004.

Lang DM. 2000. Net nearest neighbor analysis (NNNA) summarizes non-compensated dinucleotides within gene sequences. *Bioinformatics*. Mar;16(3):212-221, DOI: 10.1093/bioinformatics/16.3.212.