

2008

MotiX: a computational tool for single species evaluation of short DNA sequences based on clustering and frequency biases

Kevin Riehle
University of Dayton

Follow this and additional works at: https://ecommons.udayton.edu/graduate_theses

Recommended Citation

Riehle, Kevin, "MotiX: a computational tool for single species evaluation of short DNA sequences based on clustering and frequency biases" (2008). *Graduate Theses and Dissertations*. 5176.
https://ecommons.udayton.edu/graduate_theses/5176

This Thesis is brought to you for free and open access by the Theses and Dissertations at eCommons. It has been accepted for inclusion in Graduate Theses and Dissertations by an authorized administrator of eCommons. For more information, please contact mschlangen1@udayton.edu, ecommons@udayton.edu.

MOTIX: A COMPUTATIONAL TOOL FOR SINGLE SPECIES
EVALUATION OF SHORT DNA SEQUENCES BASED
ON CLUSTERING AND FREQUENCY BIASES

Thesis

Submitted to

The College of Arts and Sciences of the
UNIVERSITY OF DAYTON

in Partial Fulfillment of the Requirements for

The Degree

Master of Science in Computer Science

by

Kevin Riehle

UNIVERSITY OF DAYTON

Dayton, Ohio

December, 2008


UNIVERSITY OF DAYTON
THESIS APPROVAL

Date: September 4, 2008

The members of the Committee approve the thesis entitled "MotiX: A Computational Tool for Single Species Evaluation of Short DNA Sequences Based on Clustering and Frequency Biases", defended September 4, 2008.


Sudhindra R. Gadagkar
(Thesis Advisor and Committee Chairperson)


Jennifer Seitzer
(Advisory Committee Member)


James P. Buckley
(Advisory Committee Member)

It is recommended that this thesis be used in partial fulfillment of the requirements for the degree of Master of Computer Science from the Department of Computer Science at the University of Dayton.


James P. Buckley
(Chair of the Department)

The committee and the University of Dayton are not liable for any use of the materials presented in this study.

© Copyright by

Kevin Riehle

All rights reserved

2008

ABSTRACT

Gene regulation is tightly controlled in eukaryotes (organisms that have their DNA encapsulated within a nucleus), principally by means of proteins called transcription factors (TF) that bind to short stretches of the DNA (called transcription factor binding sites, TFBS), typically near genes, and generate a signal regarding the expression of a given gene. While experimental identification of TFBSs is an expensive process, computational identification is also difficult since there is no known distinguishing feature that would allow their easy identification. An additional problem is that the sequence of a given TFBS is usually so short that it is likely to have many copies, only some of which are functional. In general, the likelihood of such a sequence being functional decreases with the distance from the gene it helps regulate. Based on this tendency, a positional (clustering) measure can be made for any motif relative to the location of protein coding genes. A second measure, frequency, can also be made based on the number of copies of the motif in a given area. This is indicative since a TFBS is more likely to be under purifying selection (with few copies) or positive selection (multiple copies) in the region of its influence, depending upon its function in regulating transcription – general or specific. Using these two measures, we have determined if DNA sequences of various sizes occur more or less frequently than expected by random

chance in various areas of the human genome. To determine if a chosen DNA motif has any identifiable clustering and frequency biases, we have developed a software package (MotiX) that analyzes the distribution of such sequences, by means of the following, in-house constructs:

- A database for the frequencies of 5-8mers in all chromosomes (masked and unmasked), and upstream 1000nt for all genes (in each chromosome and the entire genome)
- Computer simulation to generate synthetic chromosomes using different nucleotide sampling units
- An algorithm to analyze frequency biases using a sliding-window approach
- A database for the clustering information for all human protein coding genes
- An algorithm to analyze the clustering biases using a sliding-window approach

To my parents ...
who are always by my side
supporting my goals and to
my sister who has inspired
my continued education

I would also like to thank
my advisor who never gave
up on me as a student,
researcher, or person

TABLE OF CONTENTS

Abstract	iii
List of Figures	viii
List of Tables	ix
 Chapter 1: Introduction	 1
1.1 Frequency Database	16
1.1.1 Expected Values	16
1.2 Computer Simulation to Generate Synthetic Target Sequences	17
1.2.1 Target Sequence Generation – Random.....	18
1.2.2 Target Sequence Generation – 5-mer Roulette Wheel.....	19
1.2.3 Target Sequence Generation – 4-mer Roulette Wheel.....	22
1.2.4 Target Sequence Generation – Random 4/5-mer Roulette Wheel	23
1.2.5 Job File Creation and the Ohio Supercomputer Center.....	24
1.2.6 Arithmetic Mean	24
1.2.7 Clustering Database.....	25
1.2.8 Clustering Database for Sliding Window Approach.....	26
1.3 TRANSFAC	28
 Chapter 2: Frequency and Clustering Biases	 30
2.1 MotiX	30
2.2 Frequency Biases	31

2.3	MotiX and TRANSFAC	32
2.4	Clustering Biases	33
2.4.1	Clustering Moving Average	34
Chapter 3:	Analyzing Gene ITGA2B	36
3.1	MotiX-Frequency Results	36
3.2	Clustering Results	38
Chapter 4:	Discussion	42
4.1	Frequency Biases	42
4.2	Clustering Biases	42
References	44
Appendix A:	findMotif.c	46
Appendix B:	SRlalyzer.pl	53
Appendix C:	MotiX-clustering.vb.....	59
Appendix D:	Motix-frequency.vb	63
Vita	68

LIST OF FIGURES

Figure 1: The Distribution of CCACC Based on 500 Simulations	11
Figure 2: Location 39,823,391 on Chromosome 17	13
Figure 3: Flow Chart of the MotiX Software Package	14
Figure 4: Roulette Wheel Probabilities	20
Figure 5: Naïve Approach Frequency Ratios	37
Figure 6: Random 4/5 Approach Frequency Ratios	37
Figure 7: Screen Shot of Clustering Profile Report.....	39
Figure 8: Clustering Sliding Window Moving Average Graph	41
Figure 9: Clustering Sliding Window Moving Average Graph with Known TFBS	41

LIST OF TABLES

Table 1: Monte Carlo Distribution Results for Simulated Methods	19
Table 2: Clustering Query Algorithm	27
Table 3: Clustering Profile Report Upstream.....	27
Table 4: Clustering Profile Report Downstream	27
Table 5: TRANSFAC File.....	29
Table 6: Frequency Sliding Window Moving Average Data.....	38
Table 7: Clustering Output Bin Categorization.....	39
Table 8: Clustering Sliding Window Moving Average Data	40

CHAPTER 1

INTRODUCTION

Until recently, non-coding DNA, constituting about 95% of the human genome, was thought to contain few or no functional elements. However, recent research has been uncovering a wide variety of elements that are responsible for gene regulation. These elements are found typically near the genes they regulate but can also be found at great distances from these genes or occasionally even within the genes themselves.

Discovering these elements (which are usually very short) is best done computationally, given the enormity of the genome (a linear string of ~3.5 billion nucleotides). Such studies are usually done by looking for parts of the genome that are conserved across species. However, relying entirely on comparative genomics has problems (e.g., difficulty in reliably aligning genomic sequences from different species). Therefore, we have utilized an approach that does not rely entirely on comparative genomics, but rather, focuses on individual genomes, in an attempt to distinguish the characteristics of functional gene regulatory elements from non-functional, non-coding “junk” DNA.

Genes are discrete regions in the genome (the aggregate of DNA in a cell) that contain the code for making proteins. Each gene in the human genome is made up of code-containing sequences called exons, interspersed with non-coding sequences called introns that are spliced out when the gene is expressed. Apart from exons and introns, there are regions of the genome that determine if, when, where, and how much of a gene product (protein) needs to be produced. These gene-regulatory regions are typically upstream of each gene, in a region called the promoter. The regulatory regions in the promoters are small sequences that bind specific proteins called transcription factors in a complex manner. It is this binding of transcription factors (TFs) to the DNA sequences in the promoter region (transcription factor binding sites, TFBSs) that begins a cascade of events relating to the expression of a given gene that regulates the expression of the gene in question.

The haploid human genome has 24 chromosomes with a total of approximately 3 billion DNA base pairs (Adenine, symbolized by "A", Cytosine - C, Guanine - G, and Thymine - T). There are between 20,000 – 25,000 genes in the human genome which are responsible for producing approximately 50,000 gene products [International Human Genome Sequencing Consortium, 2001]. This gene count actually was somewhat of a surprise since it was expected that an organism as complex as a human must have many more genes. Furthermore, only about 1.5% of the genome codes for proteins, the rest being comprised of RNA genes, regulatory sequences, and introns, and vast regions of unknown function [International Human Genome Sequencing Consortium, 2004].

Depending on when, where, and how much of a particular protein is required by the body, a complex machinery is activated and the appropriate gene(s) are expressed accordingly. In order for each gene to be expressed, it goes through two distinct processes: transcription (making an RNA copy of the DNA code for the gene product) and translation (the synthesis of proteins based on the code in the RNA) to produce the gene product – a protein or polypeptide. The TFBSs in the promoter region of a gene play a crucial role in the regulation of that gene. Therefore, there is much interest in studying these TFBSs.

Since these sequences can be very small, however, and do not contain any obvious identifying features, much of the focus in the literature has been the identification of the regions. Large scale prediction of these regions is actually best handled computationally. Therefore, this has lately become a “hot” research topic in the emerging field of bioinformatics. This study characterizes known TFBSs based on clustering and frequency biases (explained below), using bioinformatics tools and methods.

Bioinformatics is a new interdisciplinary field that borrows ideas from several fields – notably Computer Science, in order to understand biological processes better, particularly in the area of genomic science. This is a rapidly growing field due to the simultaneous growth of sequencing and computing technologies, and the fact that there is free access to the sequences of entire genomes of a number of species. This free availability of

genome sequences has also made possible computational searches for TFBSs. The majority of the work done thus far on evaluating promoter regions have been done by a process called multi-sequence comparison. Multi-sequence comparison is a popular method for comparing genome sequences from two or more different species for conserved regions. The assumption here is that conserved sequences across species must perform similar function. Multi-species sequence comparisons have yielded a vast amount of data but are constricted to the particular elements that are shared between species. For example, a comparison of the human and chimp genomes reveals a conservative estimate of the similarities between their coding DNA as approximately 98.8% [The Chimpanzee Sequencing and Analysis Consortium, 2005]. Most of this similarity is simply due to common ancestry (5-6 million years) and need not necessarily signify identical function. There are many examples, especially in the medical literature, where drugs that have worked perfectly in animal models have had different, even disastrous, results in clinical trials [Elsea et al, 2002]. These differences in response to the drugs have been linked to differences in the coding or regulatory regions between the animal models and human. Therefore, it is important to find conserved regions in a given genome that are not confounded by evolutionary phylogenetic relationships. One obvious way of avoiding the confounding effects of phylogenetic relationships is to confine the study to within genomes, rather than cross-species comparisons.

Accordingly, as a first step towards discovering and studying unknown *cis*-regulatory elements in human, this study is devoted to understanding the behavior of known TFBSs

in the context of the human genome. The hypothesis of this study is that since these TFBSs perform important gene regulatory functions by permitting the binding of transcription factors, they must have a non-random distribution within the area of influence (typically, the promoter region). In turn, this means that a TFBS sequence is expected to occur in the promoter region at frequencies either greater than or less than those expected by random chance alone, depending on whether the TFBS has a general, or specific function, respectively. This hypothesis will be tested using computational tools (developed for this purpose in a suite called MotiX.)

In this study, we approach the problem by instituting a method that evaluates short DNA motifs in terms of whether they are over- or under-represented in various target sequences from the human genome (e.g., entire chromosome, and various estimates of promoter regions).

Motif discovery and analysis has recently become a very promising and prominent division of Bioinformatics, aided by the lost cost of computational sequence analysis. The development of genome sequencing (determining the genetic code) and DNA microarray analysis of gene expression (wet lab experiments) gives rise to the demand for data-mining tools. The goals of many DNA motif (e.g., TFBS) discovery and analysis programs are to successfully retrieve known functional regulatory sequences from a set of data. I describe a few of them in the following:

BioProspector [Liu et al, 2001] is a C program that uses sophisticated algorithms such as Gibbs sampling and Markov chain Monte Carlo to examine the promoter regions of genes in the same gene expression pattern group, in search of shared regulatory sequence motifs (TFBSs). This approach uses a method to determine functional elements based on the assumption that sequence conservation infers function similarity. However, the sequence conservation can be among species or within genomes. This program reports early success in identifying binding motifs in *Saccharomyces cerevisiae*, *Bacillus subtilis*, and *Escherichia coli* which have less complex regulatory regions as compared to human. While BioProspector has been shown to be successful in less complex species, it has several drawbacks that limit its usability as a discovery tool, for a fair amount of knowledge is expected from the user about the *cis*-regulatory elements. For example, users have to supply the sequences, pick the binding method of the TF, declare the length of the motif they are wishing to evaluate, supply the block gap, supply sequences that contain the motif, and know in advance whether the motif occurs on one or both strands of the supplied sequences. Clearly, the level of detailed knowledge expected from the user puts limits on its use as a discovery tool for novel TFBSs.

MEME (pronounced MEEM) (Multiple EM for Motif Elicitation) [Bailey et al, 2006], lays claim to being one of the most widely used search tools for regulatory regions in sets of biological sequences. This process also runs a multi-sequence alignment that can be single or multi-species. Similarly to BioProspector, MEME requests that users provide

sequences that are proven to be biologically related. This program suggests that users input regulatory regions of genes that are experimentally proven to be co-regulated based on wet lab work. MEME is able to run multiple sequences from one species but is unable to run whole-genome motif discovery because the motifs become “invisible” in the context of a whole genome. Similarly to BioProspector, this software product usability is hindered by the required preconditions of the algorithm.

There are several other programs on similar lines. The majority of these motif discovery methods have been restricted to sequence-sequence comparison algorithms that require the user supply sequences that fit specific requirements. The motivation behind the present study is the inability of most users to supply such information. As a first step towards developing a predictive tool, the goal in this study is to study known TFBS sequences in human in terms of their distribution across the genome.

Specifically, the objective is to determine if it possible to distinguish between the behavior of known TFBSs from the flanking regions of the DNA, in terms of over- or under-representation. We analyze promoter sequences for this purpose by quantifying over- and under-representation, when compared to a statistical expectation.

The frequencies for a TFBS of length n nucleotides in a target sequence of length N nucleotides are based on statistical expectations are derived in several different ways.

First, we take the naïve approach of an equi-probable distribution among the four nucleotides in the genome. Based on this assumption, a known TFBS is expected to occur $\frac{N}{4^n}$ times in the target sequence. In order to produce a more advanced method, in an attempt to add biological realism, we obtain the expected value as the arithmetic mean of the frequencies derived from 500 resamplings of the target sequence. This Monte Carlo approach was done using several sampling unit sizes. The first such exercise was done with a sampling unit of size 1 nucleotide. This meant that 500 “synthetic” target sequences were produced by means of resampling the target sequence, with replacement, one nucleotide at a time. The frequency of the motif in question was then computed in each of the 500 sequences and the mean determined to give the expected value for that TFBS in that target sequence. This approach gives an expected frequency for the TFBS of $\frac{N}{\sum_i \binom{b_i}{a_i}}$, where b_i is the number of times the nucleotide a_i is repeated in the TFBS ($\sum b_i = n$). It is known, however, that empirical DNA sequences have a certain structure to them in terms of the juxtaposition of the nucleotides next to each other [Bechtel et al., 2008]. This can range from a di-nucleotide structure in non-coding sequences to tri-nucleotide codon structure in exons, and finally to much larger conserved neighborhoods such as isochores (regions in a chromosome with either low or high G+C content). Since the length of the conserved regions that define the structure in a chromosome can thus be non-homogeneous across the chromosome, the size of the sampling unit in the resampling of the target sequences is not obvious.

Therefore, we used different sampling units to resample the target sequences (single nucleotides, 4-mer, 5-mer, random sizes between 1 and 5, and random sizes between 4 and 5 nucleotides). Using each of these sampling schemes, we generated 500 synthetic target sequences. A sliding window (of size 5) is run across the upstream region of a given gene (typically 1000 nucleotides; freely available for each of the 23,575 protein coding genes in the human genome from UCSC [Pruitt et al, 2001].) The frequency (observed count) of one such 5-mer in the target sequence is compared to the distribution containing the frequencies (expected counts) for the 500 synthetic replicates of the target sequence. The position of the observed count within the expected distribution quantifies the distance of the observed count to the mean of the expected count, thus quantifying the extent of over- or under-representation of each overlapping 5-mer sequence in the upstream region. TFBSs are then characterized by comparing their footprints to those of the corresponding flanking sequences in terms of over- or under-representation.

In order to determine the best possible resampling method, we have modified our sampling unit size to produce distributions that generate different measures of over- and under-representation. DNA structure is preserved in various positions and sizes. Exons contain a 3nt structure (codon) that is located within a gene, whereas CpG elements have a 2nt structure and are generally found near the starting position of a gene. A recent paper has stated that the best sampling unit size is 4nt [Bechtel et al, 2008] due to the fact

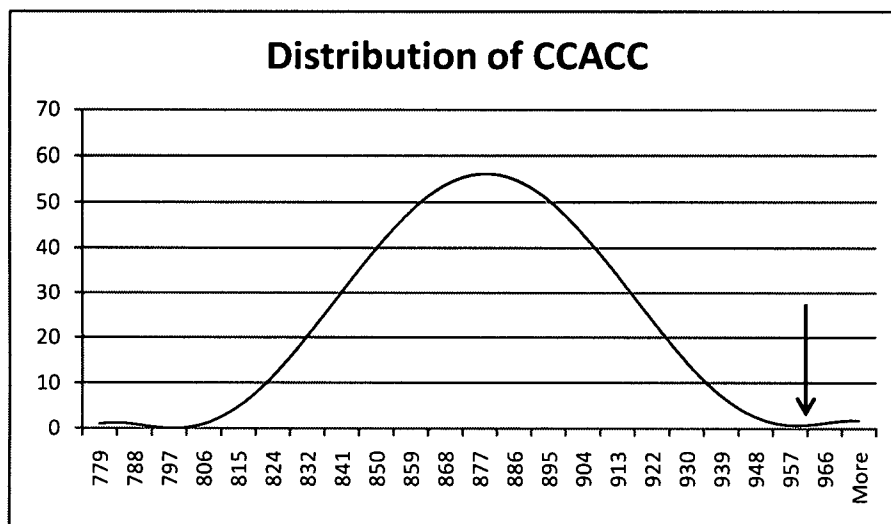
that the synthetic replicates were consistently well sampled across all of their input sequences. Table 1 displays the methods that were produced to retain the structure of the input sequences.

In order to obtain a *P*-value (the proportion of the 500 replicates that have a frequency equal to or less/greater than that of the observed frequency for the given motif) for the amount of over- and under-representation, six resampling methods were devised that produced an expected distribution. The p-value is obtained by identifying the position in the distribution (x-axis) in which our observed value fits (Figure 1). The area of the distribution from the observed position to the closest tail is calculated. This produces our *P*-value which indicates the significance of the observed hit not belonging to the expected dataset. Using the *P*-value in this manner allows us to determine the relatedness of the observed vs. the expected frequencies.

Each resampling method has the potential to produce a wide range of motifs (0-100% of the 1024-5mers) that contain measurable *P* values. An additional measurement, z-value, has also been calculated due to the fact that the *P* value could not be computed for some motifs, since the observed value lay far outside the expected distribution. The z-value

$\left(\frac{O - \mu}{\sigma} \right)$ (where *O* is the observed frequency in the target sequence, μ is the mean

Figure 1. The distribution of CCACC based on 500 simulations.



min	779
max	975
mean	875
observed	962
p	0.002

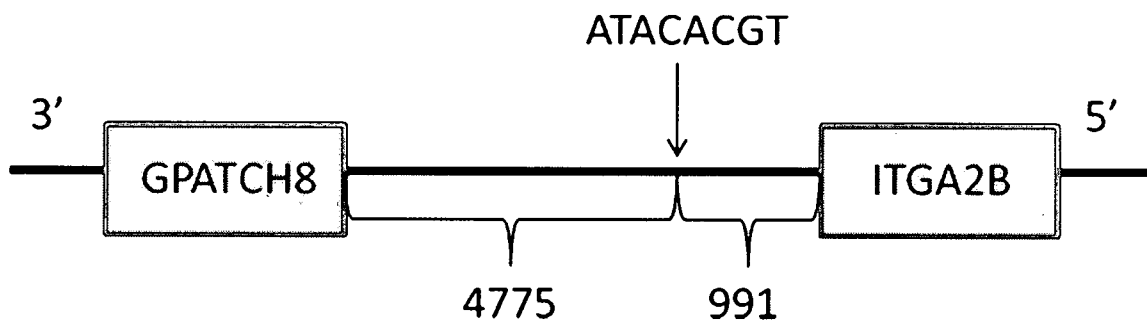
obtained from the 500 replications, and σ is the standard deviation, a measure of the variation in the expected distribution provides a statistical measurement for the analysis of each motif. It is then possible to plot and analyze the frequency biases for the entire promoter region of a gene.

Another means by which we have characterized known TFBSs is by creating a profile for each TFBS in terms of the distribution of the number of exact matches of its sequence in different regions upstream of the protein-coding gene it helps regulate. Each match is categorized based on the distance from the nearest gene upstream and downstream from the match location (Figure 2.) Figure 2 illustrates that when given a hit location within a chromosome, we can determine the distances (nt) to the nearest upstream and downstream gene loci. These locations are broken down into six bins in which we can produce a profile for each motif (0-1k, 1k-2k, 2k-5k, 5k-10k, 10k-25k, 25k-100k) that are based on the distance from the transcription start site for all genes. The first three bin sizes were chosen based on the format of the sequences provided at UCSC [Karolchik et al, 2003] (upstream 1000nt, upstream 2000nt, and upstream 5000nt) and the remaining bins were incrementally chosen to create a maximum bin distance of 100,000. Choosing a bin position to be greater than 100k increases the likelihood of producing a clustering profile that is not accurate for denser gene regions. We will generally focus on the first three bins because the majority of TFBSs that we are analyzing are rarely functional beyond this distance. Known TFBSs such as PAX6HD1 and GC box tend to have a greater concentration in the first 1000-2000nt as compared to the following 3000nt. TFBSs that exhibit a bias towards one particular bin can be considered a profile of interest.

The clustering profile algorithm addresses the needs to not only declare motifs as being over- and under-represented but to also observe positional biases that a sequence may have. Including clustering along with frequency biases addresses the complication that a

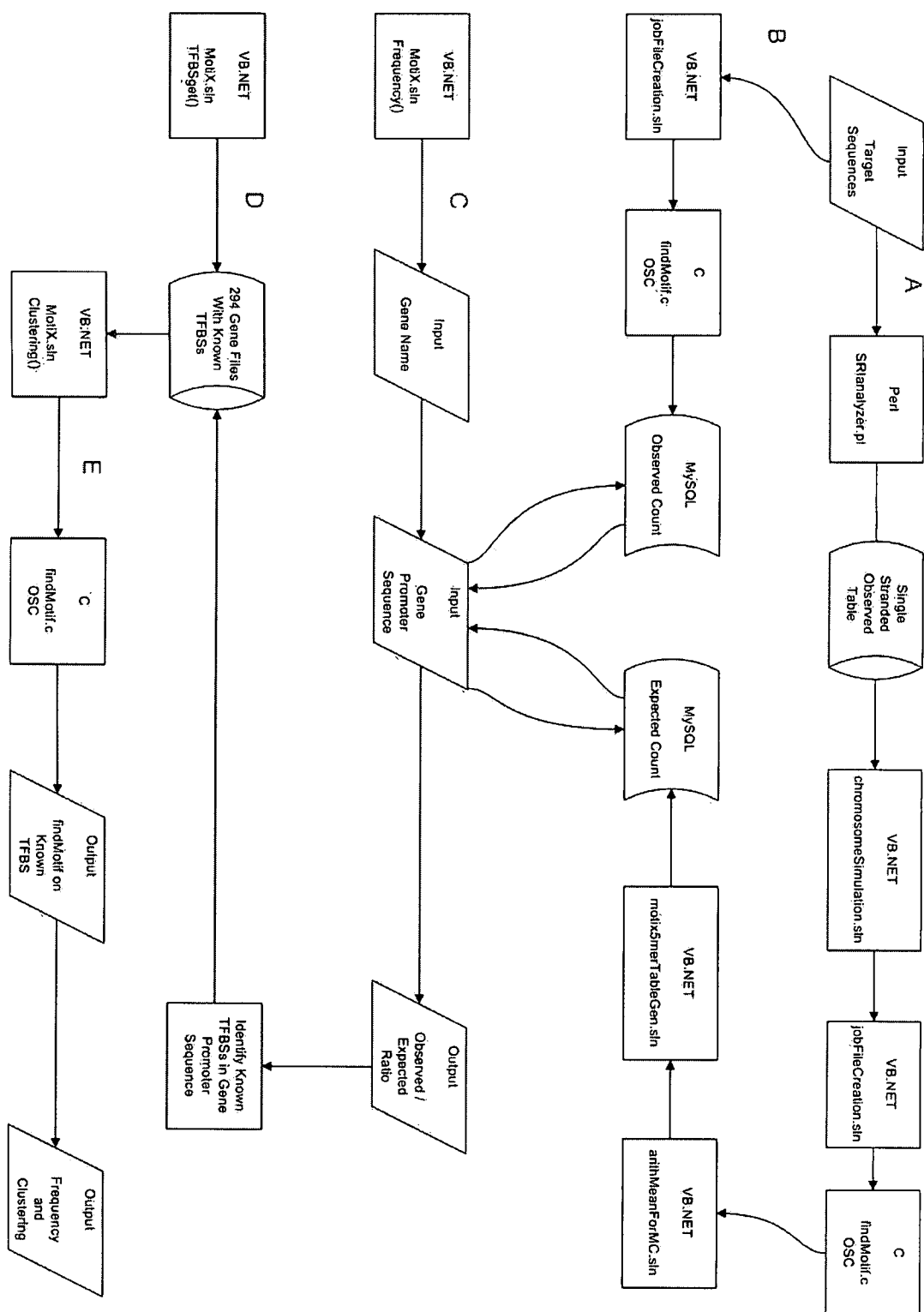
motif may occur exactly as many times as statistically expected but tends to occur more often in the vicinity of protein-coding genes. Together these algorithms are able to uniquely assign two different categories of biases.

Figure 2. Location 39,823,391 on chromosome 17 (ATACACGT).



These tasks were accomplished by means of various tools written in VB.NET, C, C++, and Perl. Figure 3 illustrates the interaction of the input files, the source code, and the output obtained from the MotiX software toolset. This process begins with the input target sequences (complete chromosomes and upstream 1000nt for each gene) obtained from UCSC (ref.) that are used in two different meaningful ways. The SRAnalyzer.pl [Bechtel et al, 2008] takes in each chromosome (Figure 3 path A) as the input and produces the observed frequencies for each 1-mer, 2-mer, 3-mer, etc. (for just the single strand.) The observed frequencies are then translated into a “.csv” file which is imported into a MySQL database. A VB.NET program (chromosomeSimulation.sln) uses the observed frequencies to create 500 simulations which are 2% of the original chromosome

Figure 3. Flow chart of the MotiX software package.



size. These simulations are then transferred to OSC (The Ohio Supercomputer Center) in which *findMotif* [Kent et al, 2002] accepts as input (initialized by creating large “.job” files via a VB.NET program (jobFileCreation.sln.)) *findMotif* outputs the frequencies of each 1024 5-mers which is transferred to a local machine. A VB.NET program (arithMeanForMC.sln) uses the *findMotif* output to create a “.csv” file that contains the arithmetic mean (the “expected” value) for each of the 1024 5-mers which is then imported into a MySQL database. A VB.NET program (motix5merTableGen.sln) utilizes combines the observed and expected values to create a “.csv” file that contains the observed, expected, variance, p-value, and z-value for each of the 1024 5-mers.

The next step in the process involves a VB.NET program (MotiX.sln – Frequency.vb) that accepts a gene name as the input and locally retrieves the appropriate promoter sequence (Figure 3 path C.) This promoter sequence is queried against the observed and expected database and retrieves the observed, expected, variance, p-value, and z-value for each 5-mer. We are then able to create a ratio comprised of the observed and expected values. The TRANSFAC file (Table 5) (text file that contains the known TFBSs for the given gene) is retrieved locally and the sequences associated with the known binding sites are labeled. These sites are isolated and sent to run on OSC (Figure 3 path E) to obtain each location in which the site occurs. This output is sent to a VB.NET program (MotiX.sln – Clustering.vb) in which the clustering profile can be obtained via the Clustering database.

1.1 Frequency Database

A database was constructed locally to house the number of exact matches for each of an exhaustive list of 1024 5-mer DNA sequences (AAAAA, AAAAC ... TTTTG, TTTTT) in each of several target sequences from the human genome: each chromosome – masked and unmasked (chromosomes sequences are available in two forms – one where all the repeat elements are “masked” and the other where the repeat sequences are spelled out), upstream 1000 bases from all protein-coding genes in the genome, upstream 1000 bases from all the protein-coding genes in the chromosome that the gene in question resides in. These exact matches were found by means of the computer program, *findMotif*, a freely available motif finding software [Kent et al, 2002]. This program allows one to specify whether the matches need to be found on only one chromosome strand or both. Since protein-coding genes are found on both strands, and therefore, cis-regulatory elements are expected to be found on both strands, we obtained the counts from both strands. These observed counts from four target sequences for each motif were entered into the database. These MySQL tables each contain two columns (motif (text) and count (integer)) and are indexed by a unique primary key (motif).

1.1.1 Expected Values

In order to determine if there is a bias in the number of matches for a given motif, we need to determine if it is over-represented or under-represented, when compared to a

statistical expectation under randomness. Due to fact that the genome is not random (the four nucleotides have unequal compositions and are distributed non-randomly in the genome) we need to develop a method that generates an appropriate expected value. Figure 1 shows the distribution of the 500 simulations of masked chromosome 17 (random 4/5 roulette wheel) for CCACC. This process is repeated for all motifs on all datasets (masked and unmasked chromosomes 1-24, promoters for chromosomes 1-24 individually, and all promoters in chromosomes 1-24 together.) This generates an expected value for each 5-mer motif on each target dataset.

1.2 Computer simulation to generate synthetic target sequences

In order to properly generate an expected value for a given motif we have developed a method that generates randomized chromosomes (masked and unmasked), upstream 1000nt for all genes, and upstream 1000nt for all the genes on the given gene's chromosome. Each of the original target sequences was resampled 500 times with replacement to generate replicate synthetic target sequences (at 2% of the original sequence length). The difficulty in this Monte Carlo approach is in choosing the appropriate sampling unit, since there is dependency among neighboring nucleotides (at the dinucleotide, tri-nucleotide levels and beyond) and this dependency is distributed randomly in the genome. Thus, the synthetic target sequences need to be generated by scrambling the original sequences, while yet preserving the appropriate dependency among neighboring nucleotides. It was attempted to reach this goal by means of the

following resampling methods: random nucleotide selection, 5-mer roulette wheel, 4-mer roulette wheel, random 1-5mer roulette wheel, random 4-5mer roulette wheel, random 4-5mer roulette wheel with SRI analyzer, and 4-mer roulette wheel with SRI analyzer.

1.2.1 Target sequence generation using randomly resampled single nucleotides

In this method, we selected a nucleotide, with replacement, from a random position in the target sequence until 2% of the total size was sampled. *findMotif* was then run on each of the 500 synthetic DNA sequences to produce as many counts for each of the 1024 5-mers. The observed value for the given motif was then transformed (divided by 50 due to the sampling size of 2%) and plotted on the distribution of the 500 data points to determine if the motif (5-mer) in question was over-represented, under represented or at statistical expectation (Figure 1.) This method produced simulations that retained the original base composition. However, only 42 out of 1024 5-mers were found to lie within the distribution of expected counts (Table 1). Thus, this method clearly did not retain the non-randomness among neighboring nucleotides that is inherent in the original target sequences. Therefore, we decided to change the sampling unit to multiple nucleotides rather than a single randomly chosen nucleotide.

Table 1. This table displays the number of 5-mers that fit within the Monte Carlo distribution for each method

Method	
Naïve	N/A
Random Character	42 / 1024
5-mer Roulette	1024 / 1024
Random 1-5mer Roulette	158 / 1024
Random 4-5mer Roulette	892 / 1024
Random 4-5mer Roulette SRI	1021 / 1024
4-mer Roulette SRI	805 / 1024

1.2.2 Target sequence generation using a 5-mer roulette wheel

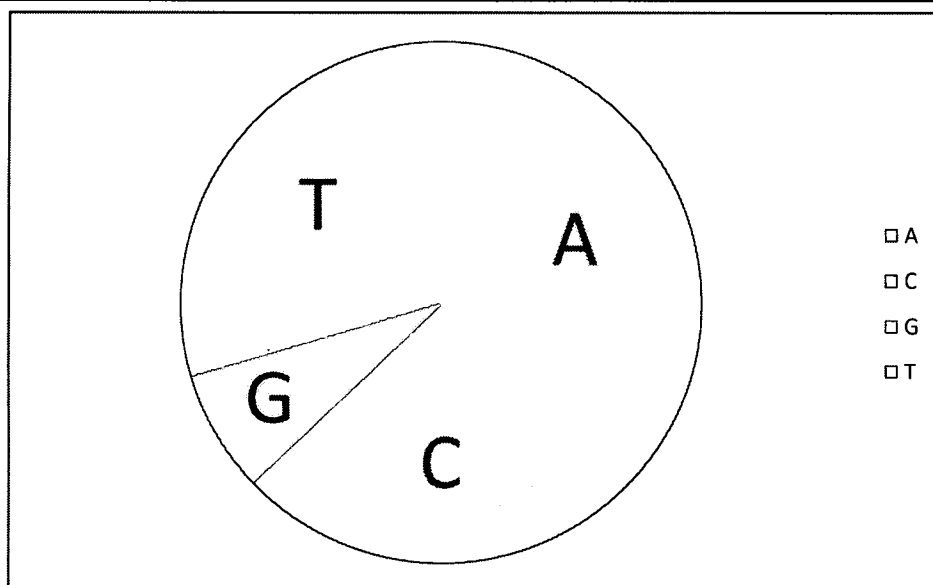
First, in order to retain the 5-mer structure in the target sequence, we resampled with a sampling unit of 5 consecutive nucleotides. This was the obvious next step due to the fact that we were evaluating sequences based on the motif length of 5. This method produced target sequences that were not randomly selected but were built based on the 5-mer frequencies in the original target sequence. This approach was adopted in light of creating sequences that would exhibit similar structural qualities to the original data sets. The first 5-mer output was generated based on the idea of a roulette wheel that has an entry for each of the 1024 5-mers and the width of each portion of the wheel is dependent on the number of exact matches in the original target sequence (observed values.) A random number was then generated between 1 and the sum of all 5-mers and based on

the position on the wheel the random number fell, the corresponding 5-mer was selected.

This formed the first five bases of a given replicate. From here on out the sequence

Figure 4. Roulette wheel probabilities of getting the next nucleotide of length 4.

sequence	next nt	motif	observed	roulette wheel range		probability	
... ATGCGT GAC	+ A	GACA	161599	1	161599	0.3681	
... ATGCGT GAC	+ C	GACC	114259	161600	275858	0.2603	
... ATGCGT GAC	+ G	GACT	33237	275859	309095	0.0757	
... ATGCGT GAC	+ T	GACG	129916	309096	439011	0.2959	
sum			439011				



would grow by one character at a time. The next character was based on the previous four characters. That is, we computed the probability of getting the previous four characters plus an A, C, T, or G, and thus chose the next nucleotide. Thus, the entire synthetic sequence was generated, and the process repeated 500 times to obtain all the replicate sequences. This method, however, generated expected values that were very

close to the observed values (Table 1) – a fact that a little mathematical reasoning showed to be expected: when the original target sequence is resampled using 5-mer sequences as the sampling unit, $P(E = O) \rightarrow 1$, as $n \rightarrow \infty$, where E is the mean (expected value) number of exact matches among the replicates, of any given 5-mer sequence, O is the observed value for that 5-mer sequence in the original target sequence, and n is the number of replicates. Clearly, then, resampling by 5-mers was incorrect.

A recent paper [Bechtel et al, 2008] found that $k = 4$ was the oligonucleotide size that was the most sampled among the sequences they studied from the human genome (that is, the observed frequencies of all possible k -mer sequences were the most evenly distributed when $k = 4$; $1 \leq k \leq 9$). Therefore, we decided to resample by 4-mers, as the Bechtel study appeared to suggest that such resampling would preserve structure in the synthetic target sequences. This was done with the help of SRI-Analyzer [Bechtel et al, 2008].

The Short-Range Inhomogeneity (SRI) Analyzer is a freely available open source program that determines the number of k -mer matches in a given target sequence. Unlike *findMotif*(ref), however, SRI-Analyzer does not determine the matches for any specific motif but does so for each of every possible k -mer in a given DNA sequence ($1 \leq k \leq 9$). Also, it only analyzes the single strand of the target sequence given to it as input data. This program was used in place of *findMotif* due to the fact that it is able to generate

frequencies for motifs that are $< 5\text{nt}$ in size. The SRI analyzer was also adopted (in place of *findMotif*) due to the fact that generating single stranded sequences should appropriately be simulated using the observed values from the single stranded SRI analysis (rather than the double stranded observed value from *findMotif*.)

1.2.3 Target sequence generation using a 4-mer roulette wheel

This simulation was done using the expected values for 4-mers (Figure 4) in an identical fashion as with the 5-mer roulette wheel, except that the observed frequencies were obtained using the SRI Analyzer. However, this exercise did not preserve the structure in terms of the 5-mer oligonucleotide frequencies, since the observed frequencies of 618 out of the 1024 5-mers fell outside the distribution of the corresponding 500 expected values (Table 1.) Since we are considering all possible 5-mer sequences, a statistical expectation would be that approximately 5% of them would lie in the tails of a distribution of values created from randomized distributions. Under this logic, resampling by 4-mers was not appropriate either since it produced ~40% of the values entirely outside the expected distributions. The results of the 4-mer roulette wheel indicated that the use of a static sampling unit size is not indicative of the actual structure preserved in the chromosome. Properly varying the sampling unit size during sequence simulation should produce a greater abundance of motifs that have measurable p-values.

Our next attempt was to resample by k -mers, where k varied randomly from 1 to 5. The reasoning behind this attempt was that the association among neighboring nucleotides was not necessarily at some fixed oligonucleotide size throughout the genome but varied. However, only 158 out of the 1024 5-mers were found to lie within the distribution of expected counts (Table 1).

1.2.4 Target sequence generation using a random 4/5-mer roulette wheel

Due to the fact that the 5-mer roulette wheel simulations were too conserved (since our interest was also in 5-mers) and the 4-mer roulette wheel simulations were not conserved enough, we devised a method that would randomly choose the next character based randomly on either the previous 3 (Figure 4) or previous 4 nucleotides. This method produced 964 out of 1024 motifs within the distribution of expected values (Table 1.) This is an excellent improvement, with the percentage of 5-mers within the tails of the distribution of expected values, and beyond, totaling to approximately 10%. Therefore, although there is at best a tenuous theoretical justification for using this resampling method, we decided to use it based on the results.

1.2.5 Job File Creation and The Ohio Supercomputer Center

A local VB.NET (Microsoft's Visual Basic language implemented on the .NET framework) program (jobFileCreation) was written to generate batch job files to be run at The Ohio Supercomputer Center (OSC) in Columbus, Ohio. These job files contain an exhaustive listing to run all combinations of 5-mers on the given input datasets. This "massively serial" method was used until the developers at OSC provided a MPI wrapper that would distribute the executions in a far more efficient manner. Rather than use the standard output of *findMotif* (target sequence, start position of match, stop position of match, strand, match score, etc.) we piped the output to "wc -l" which simply returns the number of matches. These values were stored in a separate text file for each 1024 5-mer sequence.

1.2.6 Arithmetic Mean

The output from the *findMotif* runs at OSC were transferred and stored locally to generate the expected (mean) values for each motif. A local VB.NET program (arithMeanForMonteCarlo) generates the mean based on the counts of the 500 simulations for each motif and outputs the motif and the mean to a ".csv" file that has been imported into a MySQL table. Another local VB.NET program (motif5merTableGen) uses the MySQL database (created by arithMeanForMonteCarlo) to collect the observed and expected for all 1024 5-mers. This program uses the

observed values from the 500 simulations to calculate the variance, standard deviation, z and p-values for each 5-mer which is then outputted to a “.csv” file. The z-value was adopted in place of the p-value due to the fact that many motifs had p-values of 0 (observed values that lay outside the expected distribution.)

1.2.7 Clustering Database

A local database was constructed to contain information about genes and their location in each chromosome (gene table). This data was collected by web mining the freely available submissions on Entrez Gene (the National Center for Biotechnology Information website [Maglott et al, 2005]). This database is queried with information unique to a match in the genome of a given DNA motif (start position, stop position, chromosome number and strand – plus or minus; DNA is double-stranded). The db returns clustering information about the given motif, that is, the distance to the upstream and downstream genes (unless the motif is contained within a gene) using the plus and minus stranded tables. The gene table contains eight columns: strand (tinytext), gi accession value (text), gene symbol (text), Refseq accession value (text), chromosome (tiny integer), start position of gene (big integer), stop position of gene (big integer), and exon count for gene (small integer). This table is indexed by the unique primary key Refseq.

A second and third database (plus and minus) was created to contain information about the intergenic regions of the plus and minus strands. These databases contain nine columns: chrom (tiny integer), upstream symbol of location (text), upstream gi accession value (medium integer), upstream refseq accession value (text), downstream symbol of location (text), downstream gi accession value (medium integer), downstream refseq accession value (text), start position (big integer), and stop position (big integer). This table is indexed by two primary keys (upstream refseq and downstream refseq) that together create a unique table entry. If the motif “hit” is not contained within a gene these tables relay the distance to the closest gene up and downstream (Figure 2 and Table 2.)

1.2.8 Clustering Database for Sliding Window Approach

An additional measurement was completed using a sliding window approach of size 8. Due to the fact that most known TFBSs are between 5-25nt we decided to use the largest realistic length for our sliding window size (there are 65,536 8-mers). The clustering biases were obtained for an exhaustive list of 8-mers for chromosome 17 masked. This dataset was obtained by generating the output of each 8-mer using *findMotif* against chromosome 17 which contains the locations of each hit. The *findMotif* output was then run against the clustering database which would output the counts for the desired bin sizes compared to each genes promoter region (0-1000, 1000-2000, etc.) in the form of a “.csv” file. These counts reflect the amount of times the given motif appeared in a

Table 2. Clustering algorithm.

Query gene table					
SELECT gi, symbol, refseq FROM gene WHERE (chrom=17 And (strand) = '-' And (39823391 >= stop And <= start));					
Is hit within a gene?					
Result: No Match.					
If no hit query minus table					
SELECT upsym, upgi, downsym, downgi, start, stop FROM minus WHERE chrom=17 And 39823391 >= start And <= stop;					
	upsym	upgi	downsym	downgi	start stop
Result:	ITGA2B	3674	GPATCH8	23131	39822400 39828174
Output:	upstream distance:	location - start	= 39823391 - 39822400	= 991	to ITGA2B
	downstream distance:	stop - location - size of motif	= 39828174 - 39823391 - 8	= 4775	to GPATCH8

particular demographic location on the chromosome, thus producing a clustering profile (Table 3 and Table 4.) A local database (clust17) was created that contains 14 columns based on the clustering profile information produced by the above method (indexed by the primary key "motif"): motif (text), total count (integer), upstream 0-1000nt count

Table 3. Clustering profile report for ATACACGT on chromosome 17 upstream.

0-1,000	1,000-2,000	2,000-5,000	5,000-10,000	10,000-25,000	25,000-100,000
6	1	7	6	30	58

Table 4. Clustering profile report for ATACACGT on chromosome 17 downstream.

0-1,000	1,000-2,000	2,000-5,000	5,000-10,000	10,000-25,000	25,000-100,000
4	3	6	10	25	63

(integer), upstream 1000-2000nt count (integer), upstream 2000-5000nt count (integer), upstream 5000-10,000nt count (integer), upstream 10,000-25,000nt count (integer), upstream 25,000-100,000nt count (integer), downstream 0-1000nt count (integer), downstream 1000-2000nt count (integer), downstream 2000-5000nt count (integer), downstream 5000-10,000nt count (integer), downstream 10,000-25,000nt count (integer), and downstream 25,000-100,000nt count (integer.)) This database was used to evaluate each of the 993 8-mer clustering profiles in a gene's 1000nt promoter region.

1.3 TRANSFAC

TRANSFAC is a freely available database that stores experimentally proven TFBSs [Heinemeyer et al, 1999]. We have web mined this dataset and collected information for 294 out of the 23,575 human genes [Pruitt et al, 2001]. The number of genes collected was limited by the amount of TFBSs that had confirmed sequences. These genes are associated with at least one known and experimentally proven TFBS sequence within the gene's promoter (Table 5.) This is an essential step in using experimentally proven sequences that are functional in a particular area of the genome.

Table 5 represents the data stored on the TRANSFAC web site for NM_000419. The sequence is composed of upper and lower case characters that represent the consensus sequence as well as the potential degenerate parts of the sequence, respectively.

Table 5. TRANSFAC file for NM_000419.

RO	Sequence	Name	Start	Stop
R03302	aataTGGCTGGttg	HNF-1alphaA, B, C	-562	-530
R03303	ggtCCTAGAAGgag	none	-524	-502
R03304	tcaggttTTATCGggggcagc	GATA-1	-476	-456
R08205	GGAGATTAGA	GATA-1	-381	-372
R03305	AGGCTAGAGTAGA	HNF-1alphaA, B, C	-362	-338
R08206	ATTGATAGGC	GATA-1	-249	-240
R03306	agaaCCAATag	none	-243	-220
R00596	CACCC	CACCC-binding	around	-145
R01710	tTGATAAGaa	GATA-1	-69	-39
R01711	aaagacTTCCTGTGGAGGAAtctga	c-Ets-1	-48	-24

CHAPTER 2

FREQUENCY AND CLUSTERING BIASES

In this chapter we will address the computational tools that have been developed to analyze TFBS sequences from TRANSFAC for over/under representation (which we term frequency biases). An additional measurement, clustering, has also been developed to determine the positional bias a motif may have in a given genome with respect to gene promoter regions.

2.1 MotiX

MotiX is a software package written in VB.NET for the purposes of evaluating both frequency and clustering biases. MotiX is a collective term used specifically for the operations that carry out the clustering and frequency calculations for a given gene (Figure 3 path C, E.) The remaining paths of the flowchart (Figure 3 path A, B, and D) are separate programs that have been developed to create the input files and databases necessary for MotiX, which are run once. MotiX is a VB.NET program that has two separate GUIs (clustering.vb and frequency.vb) that are currently stand alone operations.

VB.NET was chosen because it is a higher level language that can access MySQL tables, mine HTML pages, and utilize built in string parsing and array manipulation libraries in a very straight forward manner. Also, this language has continued to be employed due to the ease at which Biology undergraduates can conduct research using an easy to follow GUI.

2.2 Frequency Biases

The frequency biases portion of MotiX takes a gene name (ID) as the input for the program. The gene name is queried against the upstream 1000 dataset (ref), and the file containing the sliding window of size 5 for the promoter region of the gene is retrieved and stored locally on the machine containing the MotiX executable. A sliding window of size 5 has been constructed for the 1000 nucleotides which produces 996 5-mers ($n = \text{length} - 5 + 1$.) Each 5-mer is then queried against the observed and expected databases (previously mentioned) for the counts on host chromosome masked and unmasked, upstream 1000 for all genes and upstream 1000 for the gene's host chromosome. We then can divide the observed by the expected producing a frequency ratio that exhibits under- and over-representation for each target sequence. This dataset also includes the variance, p-value, and z-value for each of the motifs contained in the input file. As previously mentioned, we are currently giving more focus on the z-value.

2.3 MotiX and TRANSFAC

MotiX queries the TRANSFAC database to locate the proper TRANSFAC file that will be used to characterize each experimentally proven TFBS for the given gene (Table 5.) This input file contains the RO value (Accession value), promoter name, promoter sequence, and a range indicating where you will find the sequence (different wet lab results have produced slightly different locations.) Once the input file is opened, MotiX reads in each TFBS and breaks down the sequence into 5-mers (similar to the sliding window approach) and stores these values into a string array. The first 5-mer of the TFBS is queried against the sliding window array until there is a match. After the first 5-mer is correctly matched the next array position is compared to the next 5-mer position of the TFBS. This process continues until a complete match has been obtained. This match location is then compared to the location range in the TRANSFAC file to verify that the match is in the correct position due to the fact that there can be multiple copies of a TFBS in a gene's promoter. An additional array is also created to contain the ratios (observed / expected), variance, p-value, and z-values (masked, unmasked, upstream all, and upstream host.) This array is then queried for the match position and the frequency data associated for each 5-mer of the motif is outputted to a report file. This process creates a complete frequency bias output for each TFBS in the TRANSFAC file for a given gene.

2.4 Clustering Biases

Each motif characterized by the frequency bias portion of MotiX is then run against its given chromosome via *findMotif*. The given TFBS located in the TRANSFAC file is queried against the chromosome (in which the gene resides) and the output for *findMotif* (not piped) contains the location of each of the TFBSs locations. This will allow the each location to be categorized based on the distances to the flanking genes (if the hit is not contained within a gene.)

The *findMotif* output is sent to the clustering portion of MotiX in which each hit is analyzed. The location of the hit is queried against the geneTable-gene database to find out whether or not the hit resides within a gene. If this query returns NULL the geneTable-plus/minus table is queried (based on the strand in which the hit resides) to find out the distances to the closest up and downstream genes. The upstream gene distance is obtained by subtracting the stop position of the closest upstream gene by the location of the hit. The downstream gene distance is obtained by subtracting the location from the hit by the start position of the closest downstream gene (and then by the size of the motif.) These values are then categorized by the user inputted bin sizes, the default being 1k, 2k, 5k, 10k, 25k, and 100k. The pre-report file is labeled with html headers that indicate which bin category (up1k, up2k, up5k, etc.) each hit is contained in for upstream and downstream values. The intermediary file containing the clustering results is reopened and used to count the occurrences of the different bin categories (up1k, up2k, up5k, etc.) The count for each bin category is outputted to the final report file (of type

“.html”) (Figure 9.) This final report file is appended with each hits distances to the up and downstream genes. These values are color coded based on the bin category (red is the closest and light yellow is the furthest away) (Figure 7.) The concatenated report allows us to evaluate the positional bias as well as examine individual hits. This is particularly useful if you want, for example, to see which gene’s promoters that “ATATCGGTACCG” occurs within.

2.4.1 Clustering Moving Average

The clustering database for the sliding window approach (clust17) was used to determine if there were any clustering biases in a given gene’s promoter region. This method incrementally takes the next 8nt in a 1000nt sequence until the last 8-mer is obtained. This produces a 993 ($1000 - 8 + 1$) line “.csv” file that contains the 14 data points associated with the clust17 table for each 8-mer. From this data we can compute the ratio of the proportions that a given motif may have in the 0-1000 vs. the 1000-5000 region (Table 7.)

Initial observations of evaluating the clustering profile for a promoter region revealed many peaks that were difficult to quantify. We decided to take a moving average of eight 8-mers to determine if there were potentially groups of 8-mers that could collectively produce an interesting clustering profile. We suspected that we could

measure the peak or trough's width to determine the potential size of interesting elements.

CHAPTER 3

ANALYZING GENE ITGA2B

Applying the clustering and frequency portions of MotiX has produced results that uniquely try to identify interesting portions of gene promoter regions. The following results display the output produced for the human gene NM_000419 (ITGA2B.)

3.1 MotiX-Frequency Results

The six different resampling methods have produced a distinct set of data that allows the researcher to visualize the effects of varying the sampling unit size. The graph produced by plotting the z-values for upstream all, upstream host, masked, and unmasked displays the differences between the naïve approach (Figure 5) and the random 4/5-mer roulette wheel (Figure 6) promoter regions.

The random 4/5-mer roulette wheel sliding window (displayed in Table 6) strengthens the argument for using the z-value score as the four columns for the p-value often returns a score of 0. The z-value appropriates return positive (over-representation) and negative values (under-representation) for the current motif in the sliding window. The smaller or larger the z-value, the greater the level of over- or under-representation.

Figure 5. Naïve approach frequency ratios for NM_000419 (-484 to -448.)

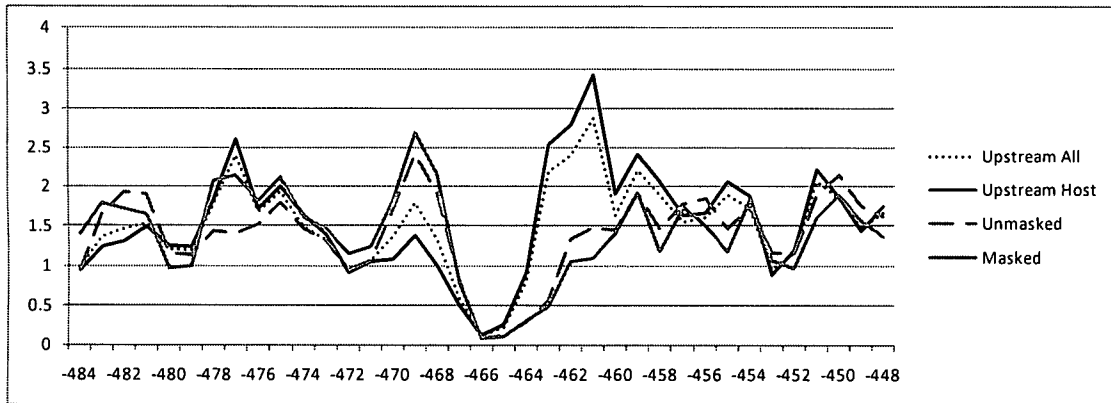


Figure 6. Random 4/5-mer approach frequency ratios for NM_000419 (-484 to -448.)

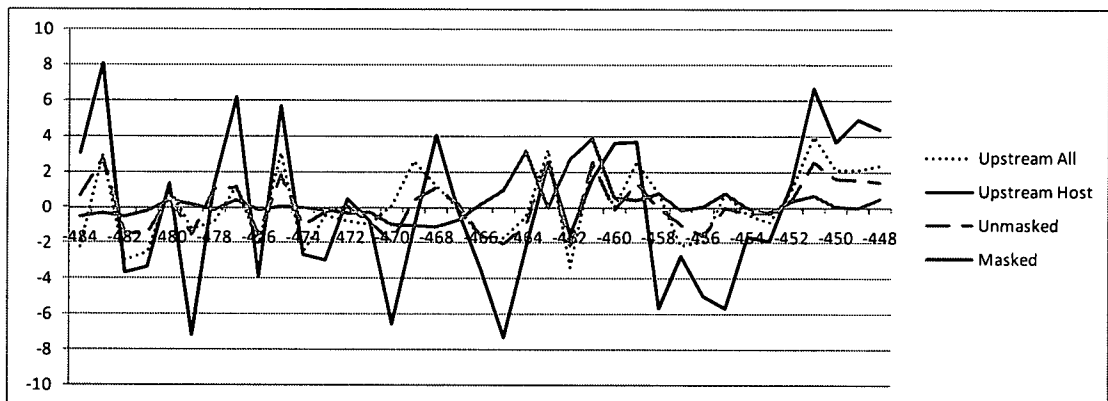


Table 6. The frequency sliding window data for NM_000419.

motif	Upstream	Upstream	Unmasked	Masked	Masked	Unmasked	Upstream	Upstream	Masked	Unmasked	Upstream	Upstream
	All	Host					Host	All			Host	All
	obs / exp	obs / exp	obs / exp	obs / exp	p	p	p	p	z	z	z	z
ataca	1.08	0.507	1.064	1.045	0.966	0	0	0.928	1.822	3.83	-0.999	1.34
tacac	0.857	0.603	0.835	0.921	0	0	0	0.01	-2.712	-8.247	-0.813	-2.829
acacg	0.919	1.37	0.931	0.98	0.322	0.012	0	0.116	-0.445	-2.201	0.58	-1.199
cacgt	0.996	1.489	0.945	0.994	0.46	0.068	0	0.492	-0.128	-1.464	0.682	-0.067
acgta	1.114	1.33	1.149	1.08	0.878	0	0	0.876	1.14	2.886	0.465	1.276
cgtac	1.079	1.829	1.066	1.02	0.58	0.844	0	0.772	0.253	1.03	1.188	1.079
gtact	0.974	0.644	0.98	0.977	0.268	0.284	0	0.37	-0.605	-0.647	-0.691	-0.426
tacta	1.127	0.532	1.135	1.073	0.982	0	0	0.952	2.129	5.705	-0.943	1.645
actac	1.007	0.724	0.966	0.967	0.168	0.07	0	0.544	-0.954	-1.429	-0.477	0.111

Plotting the input file (NM_000419) shows one particular TFBS (GATA-1) collected from TRANSFAC (sequence “tcaggttTTATCGggggcagc.”) The naïve approach (Figure 5) appears to show a brief trough for the middle section of the TFBS. The same plot for the random 4/5 roulette wheel (Figure 6) exhibits no similarities or discernable characteristics for the TRANSFAC TFBS.

3.2 Clustering Results

The clustering portion of MotiX produced a clustering profile for each 8-mer of ITGA2B's (NM_000419) promoter region. Each clustering profile is derived from a report that displays the distances of each 8-mer hit to the closest upstream and downstream gene (Table 7.) The concatenation of these results (Table 8) allow for the graph (Figure 9) of the sliding window moving average for the gene's promoter region.

Figure 7. Screen shot of clustering profile report.

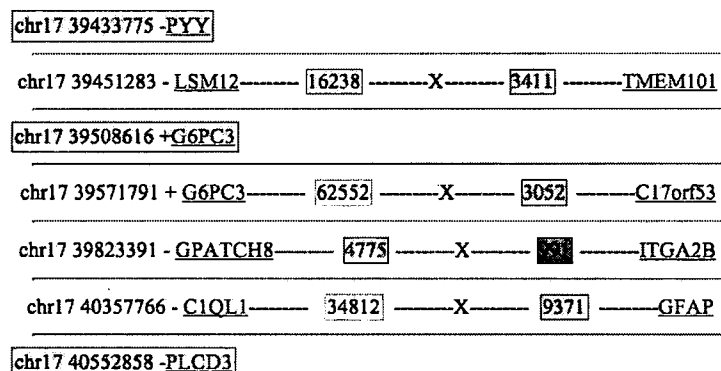


Table 7. Partial clustering output exhibiting bin categorization.

Chr.	Num.	Match Position	Str.	Gene Name	Distance to Upstream Gene	Bin Category	Distance to Downstream Gene	Gene Name	Bin Category
chr17	186	39408829	+	CD300LG	112307	none	28720	NAGS	25k - 100k
chr17	187	39433775	-	PYY	0	genic hit	0		genic hit
chr17	188	39451283	-	LSM12	16238	10k - 25k	3411	TMEM101	2k - 5k
chr17	189	39508616	+	G6PC3	0	genic hit	0		genic hit
chr17	190	39571791	+	G6PC3	62552	25k - 100k	3052	C17orf53	2k - 5k
chr17	191	39823391	-	GPATCH8	4775	2k - 5k	991	ITGA2B	0 - 1k
chr17	192	40357766	-	C1QL1	34812	25k - 100k	9371	GFAP	5k - 10k
chr17	193	40552858	-	PLCD3	0	genic hit	0		genic hit
chr17	194	40878597	+	FMNL1	198128	none	194825	C17orf69	none
chr17	195	41239782	-	*None*	38262	25k - 50k	315888	PLEKHM1	none

Calculating the sliding window moving average report for NM_000419 is incrementally done with a window size of 8 and slide size of 1 (Table 8.) This report includes the total count as well as the first three bin counts. The first 1000nt proportion (1kpro) was divided by the next 4000nt proportion (5kpro) to produce a proportion of proportions. This calculation specifically tries to identify motifs that have a bias in the first 1000 or

the next 4000. A moving average is also produced for eight 8-mers at a time to produce a moving average that yields a much smoother graph when plotted (Figure 8.)

Plotting the input file (NM_000419) shows one particular TFBS collected from TRANSFAC (sequence "tcaggttTTATCGggggcagc") that is identified by a peak in the graph (Figure 9). This plot identifies one out of the two TFBSs that are ≥ 16 nt long (Table 5.) Using a sliding window moving average of 8 allows us to search for regions in the graph that exhibit a peak ≥ 1 and have a width size of ≥ 2 (due to the fact that each data point includes the averages of the 8-mer itself along with the averages of the flanking 8-mers.) Figure 9 displays a TRANSFAC TFBS (GATA-1) that has a clustering bias spanning the width of the entire element.

Table 8. Clustering sliding window moving average report for NM_000419.

location	motif	count	0-1k count	1k-2k count	2k-5k count	1k pro	2k pro	5k pro	1kpro vs 5kpro	moving average
-672	aatctcct	1688	18	12	34	0.011	0.007	0.020	0.771	0.713
-671	atctcctt	2422	16	16	42	0.007	0.007	0.017	0.533	0.740
-670	tctccttg	3118	20	25	64	0.006	0.008	0.021	0.432	0.743
-669	ctccttgc	2358	22	17	45	0.009	0.007	0.019	0.688	0.854
-668	tccttgcc	2490	18	20	48	0.007	0.008	0.019	0.500	0.893
-667	ccttgcca	2140	25	13	38	0.012	0.006	0.018	0.974	0.984
-666	cttgccac	1532	15	9	24	0.010	0.006	0.016	0.882	0.954
-665	ttgccacc	1512	16	10	22	0.011	0.007	0.015	0.923	0.955
-664	tgccacct	2900	28	15	40	0.010	0.005	0.014	0.988	0.890

1k proportion = 1k count / count

2k proportion = 2k count / count

5k proportion = 5k count / count

1k vs 5k pro. = 1k proportion / (2k proportion + (5k proportion / 3))

moving average = average of next eight 1k vs 5k pro.

Figure 8. Clustering sliding window moving average graph for NM_000419.

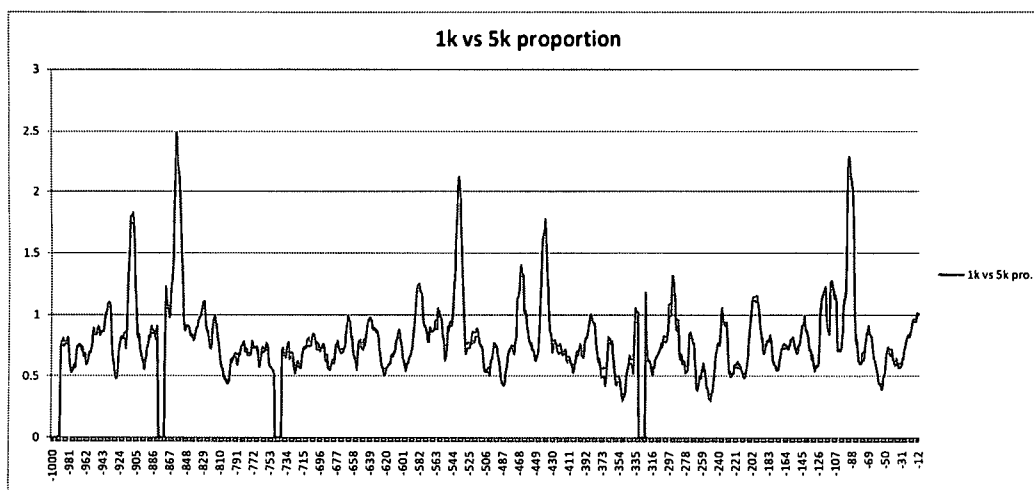
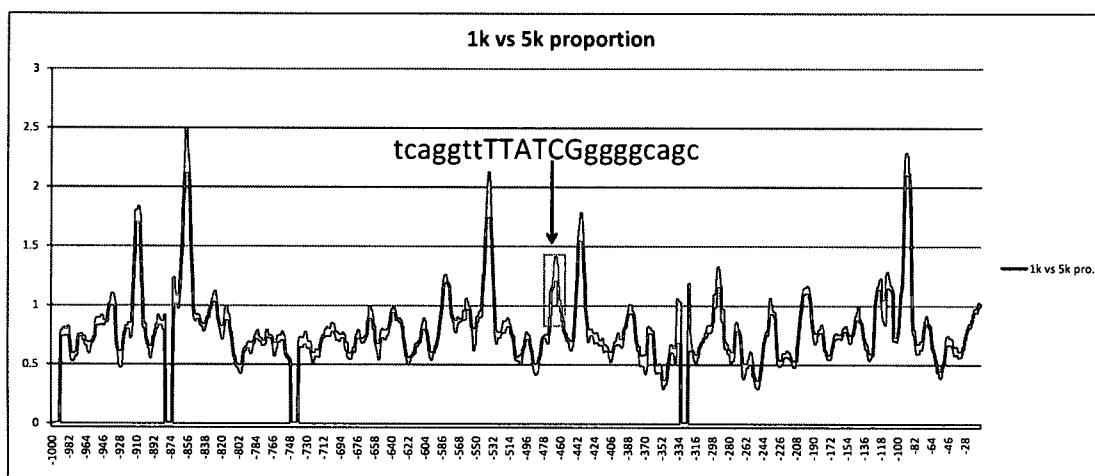


Figure 9. Clustering sliding window moving average graph for NM_000419 highlighting TFBS "tcaggttTTATCGggggcagc".



CHAPTER 4

DISCUSSION

4.1 Frequency Biases

Analyzing the graphs produced by the various sampling methods currently provide little help in the identification of regions that differ from their respective flanking regions. Each sampling method produces a chart that is visually and numerically different from the other methods. We currently have at best a tenuous theoretical justification for the use of random 4/5 roulette wheel in that it produced the greatest quantify of measurable p-values. This does not, however, suggest that we believe that it is the best sampling method. Future work in this area will be based on evaluating simulations derived from a single gene promoter. Isolating the structure of a given promoter will likely increase the probability of identifying regions that have been subject to evolutionary pressure.

4.2 Clustering Biases

The sliding window moving average graph currently produces the most exciting results obtained from this project. The clustering biases (viewed in Figure 9) exhibit the

characteristics desired from a gene promoter in that there are regions that differ from their respective flanking regions. This verified our hypothesis that regions specific to a gene promoter could potentially possess characteristics that could be examined by quantifying the clustering biases in a sliding window moving average method. Future work in this area will focus on evaluating known TFBSs (in a gene promoter) in context of its own gene (rather than include data from the entire chromosome.)

The approach in the MotiX software package developed in this study is a unique method for the identification of regions that possess a significant clustering bias. The MotiX software addresses the need for an algorithm that computationally analyzes the sequences in promoter regions that possess a significant positional bias among genic promoters. In this study the MotiX software package was used to identify a proven TFBS by analyzing the promoter region for unique clustering bias segments. While more work needs to be done, we believe that the MotiX software package can be modified to characterize many unique regions in the human genome that have not been fully qualified.

REFERENCES

Print Publications:

1. Bailey TL, Williams N, Misleh C, Li WW (2006) MEME: discovering and analyzing DNA and protein sequence motifs. *Nucleic Acids Res* 34: W369–373.
2. Bechtel, J., Wittenschlager, T., Dwyer, T., Song, J., Arunachalam, S., Ramakrishnan, S., Shepard, S., and Fedorov, A. "Genomic mid-range inhomogeneity correlates with an abundance of RNA secondary structures." *BMC Genomics*. 2008; 9: 284.
3. Elsea SH, Lucas RE. 2002. "The mousetrap: What we can learn when the mouse model does not mimic the human disease." *Institute for Laboratory Animal Research (ILAR) Journal* 43(2): 66-79.
4. Heinemeyer, T., Chen, X., Karas, H., Kel, A. E., Kel, O. V., Liebich, I., Meinhardt, T., Reuter, I., Schacherer, F. and Wingender, E. Expanding the TRANSFAC database towards an expert system of regulatory molecular mechanisms. *Nucleic Acids Res.* 27, 318-322 (1999).
5. International Human Genome Sequencing Consortium (2001). "Initial sequencing and analysis of the human genome." *Nature* 409 (6822): 860-921.
6. International Human Genome Sequencing Consortium (2004). "Finishing the euchromatic sequence of the human genome." *Nature* 431 (7011): 931-45.
7. Karolchik, D., Baertsch, R., Diekhans, M., Furey, T.S., Hinrichs, A., Lu, Y.T., Roskin, K.M., Schwartz, M., Sugnet, C.W., Thomas, D.J., Weber, R.J., Haussler, D. and Kent, W.J. The UCSC Genome Browser Database. *Nucl. Acids Res* **31**(1), 51-54 (2003).

8. Kent, W.J. 2002. BLAT—The BLAST-like alignment tool. *Genome Res.* **12**: 656–664.
9. Liu X, Brutlag DL, Liu JS. BioProspector: discovering conserved DNA motifs in upstream regulatory regions of co-expressed genes. *Pac Symp Biocomput.* 2001;:127-38.
10. Maglott D., Ostell J., Pruitt K.D., Tatusova T. Entrez Gene: gene-centered information at NCBI. *Nucleic Acids Res.* 2005;33:D54–D58.
11. The Chimpanzee Sequencing and Analysis Consortium (2005). "Initial sequence of the chimpanzee genome and comparison with the human genome." *Nature* 437 (04072).
12. Pruitt, K.D. and Maglott, D.R. 2001. RefSeq and LocusLink: NCBI gene-centered resources. *Nucleic Acids Res.* 29: 137–140.

APPENDIX A

findMotif.c

```

/* findMotif - find specified motif in nib files. */
/* written by Jim Kent's lab at UCSC */
#include "common.h"
#include "linefile.h"
#include "hash.h"
#include "options.h"
#include "dnutil.h"
#include "dnaseq.h"
#include "dnaLoad.h"
#include "portable.h"
#include "memalloc.h"
#include "obscure.h"

static char const rcsid[] = "$Id: findMotif.c,v 1.11 2006/07/27
22:39:28 hiram Exp $";

char *chr = (char *)NULL; /* process the one chromosome listed
*/
char *motif = (char *)NULL; /* specified motif string */
unsigned motifLen = 0; /* length of motif */
unsigned long long motifVal; /* motif converted to a number */
unsigned long long complementVal; /* - strand complement */
boolean bedOutput = TRUE; /* output bed file (default) */
boolean wigOutput = FALSE; /* output wiggle format instead of bed
file */
char *strand = (char *)NULL;
boolean doPlusStrand = TRUE; /* output bed file instead of wiggle
*/
boolean doMinusStrand = TRUE; /* output bed file instead of wiggle
*/

void usage()
/* Explain usage and exit. */
{
errAbort(
    "findMotif - find specified motif in sequence\n"
    "usage:\n"
    "    findMotif [options] -motif=<acgt...> sequence\n"
    "where:\n"
    "    sequence is a .fa , .nib or .2bit file or a file which is a list
of sequence files.\n"
    "options:\n"
    "    -motif=<acgt...> - search for this specified motif (case ignored,
[acgt] only)\n"
    "    -chr=<chrN> - process only this one chrN from the sequence\n"
    "    -strand=<+|-> - limit to only one strand. Default is both.\n"
    "    -bedOutput - output bed format (this is the default)\n"
    "    -wigOutput - output wiggle data format instead of bed file\n"

```

```

    "    -verbose=N - set information level [1-4]\n"
    "    NOTE: motif must be longer than 4 characters, less than 17\n"
    "    -verbose=4 - will display gaps as bed file data lines to stderr"
);
}

static struct optionSpec options[] = {
    {"chr", OPTION_STRING},
    {"strand", OPTION_STRING},
    {"motif", OPTION_STRING},
    {"bedOutput", OPTION_BOOLEAN},
    {"wigOutput", OPTION_BOOLEAN},
    {NULL, 0},
};

static void scanSeq(struct dnaSeq *seq)
{
    int i, val;
    FILE *nf = NULL;
    DNA *dna;
    unsigned long long mask;
    unsigned long long chromPosition = 0;
    register unsigned long long incomingVal = 0;
    unsigned long long incomingLength = 0;
    unsigned long long posFound = 0;
    unsigned long long negFound = 0;
    unsigned long long posPreviousPosition = 0;
    unsigned long long negPreviousPosition = 0;
    register unsigned long long posNeedle = motifVal;
    register unsigned long long negNeedle = complementVal;
    unsigned long long enterGap = 1;
    unsigned long long gapCount = 0;

    mask = 3;
    for (i=1; i < motifLen; ++i )
        mask = (mask << 2) | 3;
    verbose(2, "#\tsequence: %s size: %d, motifMask: %#llx\n", seq->name,
    seq->size, mask);
    verbose(2, "#\tmotif numerical value: %llu (%#llx)\n", posNeedle,
    posNeedle);

    /* Need "chrom" */

    dna = seq->dna;
    for (i=0; i < seq->size; ++i)
    {
        ++chromPosition;
        val = ntVal[(int)dna[i]];
        switch (val)
        {
            case T_BASE_VAL:
            case C_BASE_VAL:
            case A_BASE_VAL:
            case G_BASE_VAL:

```

```

incomingVal = mask & ((incomingVal << 2) | val);
if (! incomingLength)
{
    if ((chromPosition - enterGap) > 0)
    {
        ++gapCount;
        verbose(3,
            "\treturn from gap at %llu, gap length: %llu\n",
            chromPosition, chromPosition - enterGap);
        verbose(4, "#GAP %s\t%llu\t%llu\t%llu\t%llu\t%s\n",
            seq->name, enterGap-1, chromPosition-1, gapCount,
            chromPosition - enterGap, "+");
    }
}
++incomingLength;

if (doPlusStrand && (incomingLength >= motifLen)
    && (incomingVal == posNeedle))
{
    ++posFound;
    if (wigOutput)
        printf("%llu 1 %#llx == %#llx\n", chromPosition-
motifLen+1, incomingVal&mask, posNeedle);
    else
        printf("%s\t%llu\t%llu\t%llu\t%d\t%s\n", seq->name,
chromPosition-motifLen, chromPosition, posFound+negFound, 1000, "+");

    if ((posPreviousPosition + motifLen) > chromPosition)
        verbose(2, "#\toverlapping + at: %s:%llu-%llu\n", seq-
>name, posPreviousPosition, chromPosition);
    posPreviousPosition = chromPosition;
}

if (doMinusStrand && (incomingLength >= motifLen)
    && (incomingVal == negNeedle))
{
    ++negFound;
    if (wigOutput)
        printf("%llu -1 %#llx == %#llx\n", chromPosition-
motifLen+1, incomingVal&mask, negNeedle);
    else
        printf("%s\t%llu\t%llu\t%llu\t%d\t%s\n", seq->name,
chromPosition-motifLen, chromPosition, posFound+negFound, 1000, "-");

    if ((negPreviousPosition + motifLen) > chromPosition)
        verbose(2, "#\toverlapping - at: %s:%llu-%llu\n", seq-
>name, negPreviousPosition, chromPosition);
    negPreviousPosition = chromPosition;
}
break;

default:
    if (incomingLength)
    {

```

```

        verbose(3, "#\tenter gap at %llu\n", chromPosition);
        enterGap = chromPosition;
    }
    incomingVal = 0;
    incomingLength = 0;
    break;
}
}
if ((chromPosition - enterGap) > 0)
{
    ++gapCount;
    verbose(3,
        "\treturn from gap at %llu, gap length: %llu\n",
        chromPosition+1, chromPosition - enterGap + 1);
    verbose(4, "#GAP %s\t%llu\t%llu\t%llu\t%llu\t%s\n",
        seq->name, enterGap-1, chromPosition, gapCount,
        chromPosition - enterGap + 1, "+");
}

verbose(2, "#\tfound: %llu times + strand, %llu times - strand\n",
    posFound, negFound );
verbose(2, "#\t%% of chromosome: %g %% + strand %g %% - strand\n",
    (double) (posFound*motifLen)/(double)chromPosition,
    (double) (negFound*motifLen)/(double)chromPosition);

carefulClose(&nf);
}

static void findMotif(char *input)
/* findMotif - find specified motif in sequence file. */
{
    struct dnaLoad *dl = dnaLoadOpen(input);
    struct dnaSeq *seq;

    while ((seq = dnaLoadNext(dl)) != NULL)
    {
        verbose(2, "#\tprocessing: %s\n", seq->name);
        scanSeq(seq);
    }
}

int main(int argc, char *argv[])
/* Process command line. */
{
    int i;
    char *cp;
    unsigned long long reversed;
    size_t maxAlloc;
    char asciiAlloc[32];

    optionInit(&argc, argv, options);

    if (argc < 2)
        usage();

```

```

maxAlloc = 2100000000 *
    (((sizeof(size_t)/4)*(sizeof(size_t)/4)*(sizeof(size_t)/4)));
sprintfLongWithCommas(asciiAlloc, (long long) maxAlloc);
verbose(4, "#\tmaxAlloc: %s\n", asciiAlloc);
setMaxAlloc(maxAlloc);
/* produces: size_t is 4 == 2100000000 ~= 2^31 = 2Gb
 *           size_t is 8 = 16800000000 ~= 2^34 = 16 Gb
 */

dnaUtilOpen();

motif = optionVal("motif", NULL);
chr = optionVal("chr", NULL);
strand = optionVal("strand", NULL);
bedOutput = optionExists("bedOutput");
wigOutput = optionExists("wigOutput");

if (wigOutput)
    bedOutput = FALSE;
else
    bedOutput = TRUE;

if (chr)
    verbose(2, "#\tprocessing chr: %s\n", chr);
if (strand)
    verbose(2, "#\tprocessing strand: '%s'\n", strand);
if (motif)
    verbose(2, "#\tsearching for motif: %s\n", motif);
else {
    warn("ERROR: -motif string empty, please specify a motif\n");
    usage();
}
verbose(2, "#\tttype output: %s\n", wigOutput ? "wiggle data" : "bed
format");
verbose(2, "#\tspecified sequence: %s\n", argv[1]);
verbose(2, "#\tsizeof(motifVal): %d\n", (int)sizeof(motifVal));
if (strand)
{
    if (! (sameString(strand, "+") | sameString(strand, "-")))
    {
        warn("ERROR: -strand specified ('%s') is not + or - ?\n",
strand);
        usage();
    }
    /*      They are both on by default, turn off the one not specified
*/
    if (sameString(strand, "-"))
        doPlusStrand = FALSE;
    if (sameString(strand, "+"))
        doMinusStrand = FALSE;
}
motifLen = strlen(motif);
/*      at two bits per character, size limit of motif is

```

```

    *    number of bits in motifVal / 2
    */
if (motifLen > (4*sizeof(motifVal))/2 )
{
    warn("ERROR: motif string too long, limit %d\n",
(4*(int)sizeof(motifVal))/2 );
    usage();
}
cp = motif;
motifVal = 0;
complementVal = 0;
for (i = 0; i < motifLen; ++i)
{
    switch (*cp)
    {
    case 'a':
    case 'A':
        motifVal = (motifVal << 2) | A_BASE_VAL;
        complementVal = (complementVal << 2) | T_BASE_VAL;
        break;
    case 'c':
    case 'C':
        motifVal = (motifVal << 2) | C_BASE_VAL;
        complementVal = (complementVal << 2) | G_BASE_VAL;
        break;
    case 'g':
    case 'G':
        motifVal = (motifVal << 2) | G_BASE_VAL;
        complementVal = (complementVal << 2) | C_BASE_VAL;
        break;
    case 't':
    case 'T':
        motifVal = (motifVal << 2) | T_BASE_VAL;
        complementVal = (complementVal << 2) | A_BASE_VAL;
        break;
    default:
        warn(
            "ERROR: character in motif: '%c' is not one of ACGT\n",
            *cp);
        usage();
    }
    ++cp;
}
reversed = 0;
for (i = 0; i < motifLen; ++i)
{
    int base;
    base = complementVal & 3;
    reversed = (reversed << 2) | base;
    complementVal >>= 2;
}
complementVal = reversed;
verbose(2, "#\tmotif numerical value: %llu (%#llx)\n", motifVal,
motifVal);

```

```
verbose(2, "#\tcomplement numerical value: %llu (%#llx)\n",  
complementVal, complementVal);  
if (motifLen < 5)  
{  
    warn("ERROR: motif string must be more than 4 characters\n");  
    usage();  
}  
  
findMotif(argv[1]);  
return 0;  
}
```

APPENDIX B

```
#!/usr/bin/perl -w

# Perl script: NTcomposition.pl
# Authors: Jun Song, Trisha Dwyer, Sadeesh K. Ramakrishnan, Sasi
Arunachalam,
# Tom Wittenschlaeger, Jason M. Bechtel & Alexei Fedorov
# Bioinformatics Lab, Department of Medicine
# Medical College of Ohio, Toledo, Ohio, USA
# written April, 2007
# updated and expanded July, 2007 by Jason M. Bechtel
# Contact alexei.fedorov@utoledo.edu
#
# LICENSE: GPL version 3 or later
# If you use this code or incorporate it in another program, please
cite our paper:
# Bechtel JM, Wittenschlaeger T, Dwyer T, Song J, Arunachalam S,
Ramakrishnan SK, Shepard S, Fedorov A: Genomic mid-range inhomogeneity
correlates with an abundance of RNA secondary structures, BMC Genomics
2008, 9:284.
#
# This Perl script does the following:
# - Section 1 creates all possible oligonucleotide strings up to $N-
mers and initializes variables
# - Section 2 reads the input file (in FASTA format), gets the
nucleotide sequences,
# and counts their oligonucleotide composition, keeping the counts
in associative
# arrays: %nt1 for single nucleotides; %nt2 for dinucleotides;
... %ntN for N-mers
# - Section 3 counts the total number of oligonucleotides for each
associative array %ntN
# - Section 4 calculates the frequency of each oligonucleotide in the
associative array
# - Section 5 prints the frequency of each oligonucleotide to the
output file
# o This output is suitable for generating sequences using
"prog_rand_seq".
# - Section 6 prints the most and least common oligonucleotides to
another output file
# This version of the script attempts to handle non-canonical positions
(eg: X,N,R,Y) rationally.
# It identifies the overall canonical and non-canonical positions,
splits up each sequence at any
# non-canonical positions/runs, and then analyses the resulting sub-
sequences. The overall
# canonical vs. non-canonical content is added to the main output file
and a warning is written
# to STDERR.
#
```



```

# TODO: Handle *all* non-canonical "letters" (eg: '.', 'n', etc.)
#

if (scalar(@ARGV) != 3) {
    print STDOUT "Usage: NTcomposition.pl <infile.fa> <outfile>
<oligo_max>\n";
    print STDOUT "\twhere <infile.fa> is the name of a file containing
sequences to be analyzed (in FASTA format),\n";
    print STDOUT "\t<outfile> is the name of a file to hold the
output,\n";
    print STDOUT "\tand <oligo_max> is the longest oligonucleotide size
to be analyzed (1 to 9)\n";
    print STDOUT "\n";
    exit 1;
}

$infile = $ARGV[0]; # name of file containing sequences to analyze (in
FASTA format)
$outfile = $ARGV[1]; # name of file to write resulting oligonucleotide
frequencies to
$outfile2 = $ARGV[1] . ".tbl"; # name of file to write resulting
oligonucleotide frequencies to
$N = int( $ARGV[2] ); # longest oligonucleotide size to appear in the
output table
# Check that $N is an integer between 1 and 9 (10 and more take a lot
of memory)
if ( $N != $ARGV[2] ) { die " > Error: Invalid oligonucleotide length
[$ARGV[2]]; <N> must be an integer (and 9 >= N >= 2)\n"; }
if ( $N < 1 || $N > 9 ) { die "Invalid oligonucleotide length
[$ARGV[2]]; (9 >= N >= 2)\n"; }

# flush on every write
select(STDOUT); $| = 1;

# Section 1 creates all possible oligonucleotide strings up to $N-mers
and initializes variables
# Define the alphabet
$alphabet = "ATCG";
print STDOUT "Initializing...";
@letters = split( '', $alphabet );
# Generate all possible n-mers in this alphabet...
@new = @letters;
for $n ( 1 .. $N ) {
    @{ 'NT' . $n } = @new;
    @new = ();
    foreach $x (@letters) {
        foreach $y ( @{ 'NT' . $n } ) {
            push( @new, ( $x . $y ) );
        }
    }
}
# Initialize variables
for $n ( 1 .. $N ) {

```

```

    ${ 'total' . $n } = 0; # total # of oligos counted for each oligo
size
    %{ 'nt' . $n } = (); # oligo counts for a given oligo size
    %{ 'ntf' . $n } = (); # oligo frequencies for a given oligo size
    foreach $oligo (@{ 'NT' . $n }) {
        ${ 'nt' . $n }{ $oligo } = 0;
        ${ 'ntf' . $n }{ $oligo } = 0;
    }
}
$total_len = 0;
$total_noncanon = 0;
$overall_pct_noncanon = 0;
print STDOUT " done\n";

# Section 2 reads the input file (in FASTA format), gets the nucleotide
sequences,
# and counts their oligonucleotide composition, keeping the counts in
associative
# arrays: %nt1 for single nucleotides; %nt2 for dinucleotides; ...
%ntN for N-mers
$/ = "\n>";
open( INFILE, "$infile" ) || die "Can't open \"$infile\": $!\n";
open( OUTPUT, ">$outfile" ) || die "Can't open \"$outfile\": $!\n";
print STDOUT "Reading in sequences from \"$infile\"...";
$num_recs_read = 0;
$num_recs_processed = 0;
while (<INFILE>) {
    $num_recs_read++;
    $seq = '';
    @lines = split( "\n", $_ );
    chomp( @lines ); chomp( @lines );
    if ($num_recs_read == 1) {
        if ($lines[0] !~ /^>/) { die " does not appear to be a FASTA-
formatted file\n"; }
    }
    for $z ( 1 .. $#lines ) {
        # skip FASTA format comment lines
        if ( $lines[$z] =~ /^[>|]/ ) { next; }
        # convert to all-uppercase
        $lines[$z] = uc($lines[$z]);
        # strip out all non-letters
        # NOOOO! There are particular accepted forms of punctuation in
genomic sequences
        # and we want to characterize their presence. --jmb, 7/17/07
        $lines[$z] =~ s/[^A-Z]//g;
        # strip out all white-space
        $lines[$z] =~ s/\s//g;
        $seq .= $lines[$z];
    }
    $length = length($seq);
    $total_len += $length;
    if ( $length > 0 ) {
        $full_length = length($seq);

```

```

etc.)      # remove non-canonical bases (uncertain positions, punctuation,
           # etc.)
           @seqs = ( $seq );
           $seq =~ s/[^$alphabet]//g;
           # recalculate length
           $length = length($seq);
           $num_noncanon = $full_length - $length;
           $total_noncanon += $num_noncanon;
           # $pct_noncanon = $num_noncanon / $full_length * 100;
           if ( $num_noncanon > 0 ) {
               @seqs = split( /[^$alphabet]+/, $seqs[0] );
           }
           # for each sub-sequence (if non-canonical characters caused the
           # sequence to be split)
           foreach $seq ( @seqs ) {
               $length = length($seq);
               if ( $length > 0 ) {
                   for $n ( 1 .. $N ) {
                       if ( $length >= $n ) {
                           # count all oligos of length $n
                           for $x ( 0 .. ( $length - $n ) ) { # NOT "$length -
$length + 1"! (7/16/07 --jmb)
                               # substr offsets start at zero (0)!
                               ${ 'nt' . $n }{ substr( $seq, $x, $n ) }++;
                           }
                       }
                   }
               }
           }
           $num_recs_processed++;
       }
       close (INFILE);
       print STDOUT " $num_recs_read records found\n";
       if ($num_recs_read == 0) { die "No FASTA records in input file
[\"$infile\"]!\nAborting...\n"; }
       if ( $total_noncanon > 0 ) {
           $overall_pct_noncanon = $total_noncanon / $total_len * 100;
           printf STDERR " > Warning:  %d non-canonical positions found
comprising %.2f%% of bulk sequence data\n\n", $total_noncanon,
$overall_pct_noncanon;
       }
       print STDOUT "Calculating oligonucleotide frequencies...";

       # Section 3 counts the total number of oligonucleotides for each
       # associative array %ntN
       for $n ( 1 .. $N ) {
           foreach $mer ( @{ 'NT' . $n } ) {
               ${ 'total' . $n } += ${ 'nt' . $n }{ $mer };
           }
       }

       # Section 4 calculates the frequency of each oligonucleotide in the
       # associative array

```

```

for $n ( 1 .. $N ) {
    $z = $n + 2; # decimal places of precision for this N-mer level
    if ( ${ 'total' . $n } == 0 ) {
        foreach $mer ( @{ 'NT' . $n } ) {
            ${ 'ntf' . $n }{$mer} =
                sprintf( "%. ${z}f", 0 );
        }
    } else {
        foreach $mer ( @{ 'NT' . $n } ) {
            ${ 'ntf' . $n }{$mer} =
                sprintf( "%. ${z}f", ${ 'nt' . $n }{$mer} / ${ 'total' .
$n } );
        }
    }
}

print STDOUT " done\n";
print STDOUT "Writing frequencies to \"$outfile\"...";

# Section 5 prints the frequency of each oligonucleotide to the output
file
for $n ( 1 .. $N ) {
    @mers = @{ 'NT' . $n };
    foreach $mer ( 0 .. $#mers ) {
        print OUTPUT $mers[$mer], "\t", ${ 'ntf' . $n }{ $mers[$mer] },
"\t", ${ 'nt' . $n }{ $mers[$mer] }, "\n";
    }
    if ( $n == 1 && $total_noncanon > 0 ) {
        $total_canon = ( $total_len - $total_noncanon );
        $overall_pct_canon = $total_canon / $total_len * 100;
        $overall_pct_noncanon = $total_noncanon / $total_len * 100;
        #printf OUTPUT "[%s]\t%.3f\t%d\n", $alphabet,
$overall_pct_canon / 100, $total_canon;
        #printf OUTPUT "[^%s]\t%.3f\t%d\n", $alphabet,
$overall_pct_noncanon / 100, $total_noncanon;
        printf OUTPUT "[%s]\t%.3f\t%d\n", "N", $overall_pct_canon /
100, $total_canon;
        printf OUTPUT "[!%s]\t%.3f\t%d\n", "N", $overall_pct_noncanon /
100, $total_noncanon;
    }
    print OUTPUT "\n\n";
}
close (OUTPUT);

print STDOUT " done\n";
print STDOUT "Writing most- and least-common n-mers table to
\"$outfile2\"...";

# Section 6 prints the most and least common oligonucleotides to
another output file
open( OUTPUT, ">$outfile2" ) || die "Can't open \"$outfile2\": $!\n";
print OUTPUT "      Most- and Least-common N-mers\n\n";
print OUTPUT "\tmost\tleast\n";
for $n ( 1 .. $N ) {

```

```

    #freqs = %{ 'ntf' . $n };
    # sort the hash of frequencies by value
    #@sorted = sort { $freqs{$a} <=> $freqs{$b} } keys %freqs;
    # sorting takes too much memory for 9-mers and up... just traverse
the hash
    %counts = %{ 'nt' . $n };
    $max_count = -1;
    @max_idx = ();
    $min_count = -1;
    @min_idx = ();
    while(($mer, $count) = each(%counts)) {
        # grab the highest- and lowest-count n-mers
        if ($count > $max_count || $max_count == -1) {
            @max_idx = ();
            push(@max_idx, $mer);
            $max_count = $count;
        }
        # track all oligos with max/min count
        elsif ($count == $max_count) {
            push(@max_idx, $mer);
        }
        if ($count < $min_count || $min_count == -1) {
            @min_idx = ();
            push(@min_idx, $mer);
            $min_count = $count;
        }
        elsif ($count == $min_count) {
            push(@min_idx, $mer);
        }
    }
    print OUTPUT "$n-mers";
    print OUTPUT "\t", $counts{ $max_idx[0] };
    if (scalar(@max_idx) >= 1) {
        print OUTPUT "/";
        print OUTPUT join(',', @max_idx);
    }
    print OUTPUT "\t", $counts{ $min_idx[0] };
    if (scalar(@min_idx) >= 1) {
        print OUTPUT "/";
        print OUTPUT join(',', @min_idx);
    }
    print OUTPUT "\n";
}
close (OUTPUT);
print STDOUT " done\n";

```

APPENDIX C

```

'MotiX-clustering.vb
'Main function that analyzes a gene's promoter
'Written by Kevin Riehle

Public Sub regCheck_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles regCheck.Click
'Public Function regCheck_Click(ByVal fmotif As String, ByVal fsave As
String, ByVal ffile As String, ByVal fsize As String)

Dim motif As String = ""
Dim exList As String = "C:\Documents and Settings\Kevin_PR\My
Documents\monteCarlo\genes\NM_002026-trimmed.txt"

'open the input file containing the list of motifs in the gene promoter
Dim exListFileStream As FileStream = New FileStream(exList,
FileMode.Open, FileAccess.Read)
Dim exReader As New StreamReader(exListFileStream)

'create the output file that contains the concatenated output
Dim dbOutFile As String = saveLoc.Text + "2020.csv"
Dim dbFileStream As FileStream = New FileStream(dbOutFile,
FileMode.Create, FileAccess.Write)
Dim dbWriter As New StreamWriter(dbFileStream)

Dim exCount As Integer = 1
Dim returnVal As String = ""

Dim preLoop As Integer = 0
'read in the first line that contains the file information
motif = exReader.ReadLine()

Do
    Dim temp As String = ""
    'read in and capitalize the motif
    motif = exReader.ReadLine()
    If motif = "!" Then Exit Do
    motif = motif.ToUpper

    'create the output tile
    Dim filename3 As String = saveLoc.Text + "\" + motif
    Dim filename4 As String = filename3 + ".txt"
    Dim fss3 As FileStream = New FileStream(filename4, FileMode.Create,
FileAccess.Write, FileShare.ReadWrite)
    Dim output3 As New StreamWriter(fss3)
    Dim tempFile As String = ""

```

```

Dim statusError As String = ""

Dim completeCount As Integer = 0
output3.WriteLine(color(0, ""))

'open the input file containing the next motif
Dim fileName As String = txtFileName.Text + motif + ".txt"
Dim fs As FileStream = New FileStream(fileName, FileMode.Open,
FileAccess.Read)
Dim sw As New StreamReader(fs)

Dim Counter As Integer = 0
Dim countTotal As String = 0
'update user status bar
StatusBar1.Text = Counter.ToString + " / " + countTotal + "
" + completeCount.ToString

'get first sequence
Dim chrom As String
Dim strand As String
Dim startLoc As String
Dim stopLoc As String
Dim firstSeq As String
Dim htmlColor As String = "<table bgcolor=""00ffff""
border=""0""><tr><td>"
Dim htmlNoColor As String = "<table bgcolor=""ffffff""
border=""0""><tr><td>"
Dim htmlColorEnd As String = "</td></tr></table>"

Dim array0count As Integer = 0
Dim linecount As Integer = 0
'user option to skip X amount of lines in input file
If Not startcount.Text = 1 Then
    Do
        Dim blanks As String = sw.ReadLine()
        linecount += 1
        If linecount >= startcount.Text Then Exit Do
    Loop
End If
'read line
temp = sw.ReadLine()
Do

    'split array at space
    Dim arrWords() As String =
System.Text.RegularExpressions.Regex.Split(temp, "\s+")
    chrom = arrWords(0)
    'split at 'r' to retrieve chromosome
    Dim tempChrom() As String = chrom.Split("r")
    chrom = tempChrom(1)
    chrom = tempChrom(1)
    startLoc = arrWords(1)
    strand = arrWords(2)

```

```

If (chrom = "X") Then
    chrom = "23"
ElseIf (chrom = "Y") Then
    chrom = "24"
End If
'database function to determine if hit is within a gene
Dim dbReturn As String = dbX2(startLoc, chrom, strand)

dbReturn = dbReturn + " "
If (dbReturn.Length > 20) Then
    firstSeq = htmlColor + temp + dbReturn + "&#09;" +
htmlColorEnd
Else
    'else find out between which genes the motif is located
    dbReturn = dbStream(startLoc, chrom, strand, seqLength.Text)
    Dim tempDb As String = color(temp + " " + dbReturn, strand)
    firstSeq = tempDb
    firstSeq = firstSeq + " "
    If (firstSeq.Length < 10) Then
        statusError = "<<<<<<<ERROR>>>>>>>"
    Else
        statusError = " "
    End If
End If

'output
output3.WriteLine("<hr width=""95%"">")
output3.WriteLine(firstSeq)

Counter += 1
completeCount += 1
StatusBar1.Text = Counter.ToString + " / " + chrom + "
" + statusError
array0count += 1

'read the next line
temp = sw.ReadLine()
If (temp = "!") Or (temp = Nothing) Then Exit Do

Loop
output3.Close()
'eOutput.Close()
sw.DiscardBufferedData()
sw.Close()

'make a copy of output and send to report for final html
preparation
Dim fssReOpen As FileStream = New FileStream(filename4,
FileMode.Open, FileAccess.Read)
Dim input As New StreamReader(fssReOpen)
Dim source As String = input.ReadToEnd()
input.Close()
'if you want to delete the html report then set to true
returnVal = report(filename4, filename3, source, True)

```



```
dbWriter.WriteLine(motif + "," + returnVal)

exCount += 1
If exCount = 65537 Then Exit Do

Loop
exReader.Close()
dbWriter.Close()
MsgBox("el fin.")

End Sub
End Class
```

APPENDIX D

```

'MotiX-frequency.vb
'Written by Kevin Riehle

'main function
Private Sub run_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles run.Click

Dim refReadFile As String = transfacFILE.Text + "20from17.txt"
Dim allRefReader As FileStream = New FileStream(refReadFile,
FileMode.Open, FileAccess.Read)
Dim rRead As New StreamReader(allRefReader)

Dim refSeqLine As String
Dim globalCount As Integer = 0

Dim currMotif As String
'decimals for masked, unmasked, upstreamall
'    and upstream host values
Dim McurrExp As Decimal
Dim UcurrExp As Decimal
Dim AcurrExp As Decimal
Dim OcurrExp As Decimal

Dim line As String
Dim chromDB As String

Do
    'read in line and seperate at space to get gene name
    line = rRead.ReadLine()
    Dim refComma() As String = line.Split(" ")

    refSeqLine = refComma(0)
    chromDB = refComma(1)

    If refSeqLine Is Nothing Then Exit Do
    If refSeqLine = "!" Then Exit Do

    status.Text = refSeqLine

    'create a directory named after gene
    Try
        Directory.CreateDirectory(tempLoc.Text + refSeqLine)
    Catch
        MsgBox("error creating: " & tempLoc.Text & refSeqLine & "
directory")
    End Try
End Do

```

```

End Try
Dim checkboxVals() As String = sizeANDlength()

'uncomment this out to get files from local server
'Options()

Dim loc As String = tempLoc.Text + refSeqLine + "\" + refSeqLine
Dim maskedFile As String = loc + checkboxVals(0)
Dim unmaskedFile As String = loc + checkboxVals(1)
Dim uplkfile As String = loc + checkboxVals(2)
'open three file streams for uplk, upmasked, and upunmasked
Dim open1 As FileStream = New FileStream(maskedFile, FileMode.Open,
FileAccess.Read)
Dim mRead As New StreamReader(open1)
Dim open2 As FileStream = New FileStream(unmaskedFile,
FileMode.Open, FileAccess.Read)
Dim uRead As New StreamReader(open2)
Dim open3 As FileStream = New FileStream(uplkfile, FileMode.Open,
FileAccess.Read)
Dim aRead As New StreamReader(open3)

'open output file that will contain motif frequencies
Dim outLoc As String = loc + "-output.txt"
Dim writel As FileStream = New FileStream(outLoc, FileMode.Create,
FileAccess.Write)
Dim out As New StreamWriter(writel)

'open output file that will contain the Monte Carlo results for
each motif
Dim frequencyOutput As String = loc + "-" + checkboxVals(5) + "-
freqOUT-lmc-" + runTYPE + ".txt"
Dim write2 As FileStream = New FileStream(frequencyOutput,
FileMode.Create, FileAccess.Write)
Dim freqOut As New StreamWriter(write2)

Dim lineM As String
Dim lineU As String
Dim lineA As String
Dim lineO As String
'read in each file to retrieve observed count
lineM = mRead.ReadLine()
lineU = uRead.ReadLine()
lineA = aRead.ReadLine()
'split at 'r' to get chromosome
Dim lineSplit() As String = lineM.Split("r")
Dim chrom As String = lineSplit(1)
'get masked length by function for masked and unmasked lengths
Dim maskedMultiVals() As Decimal = maskedLength(chrom)

'for the NAIVE approach these values won't change so we can get
them once
'get masked expected
Dim expM As Decimal = maskedMultiVals(0)
'get unmasked expected

```

```

Dim expU As Decimal = unmaskedLength(chrom)
'get upstreamAll expected
Dim expA As Decimal = maskedMultiVals(4)
'get upstreamHost expected
Dim expH As Decimal = maskedMultiVals(5)

Dim count As Integer = 0
'create arrays to hold expected values for the 4 expected locations
Dim arrayM(5000) As Decimal
Dim arrayU(5000) As Decimal
Dim arrayA(5000) As Decimal
Dim arrayO(5000) As Decimal
Dim temp As Decimal
Dim arrayMOTIF(5000) As String

'need to read through X lines of host masked and unmasked to get to
the last 1000nt
' because each file contains upstream5000
Do While count < maskedMultiVals(2)
    lineM = mRead.ReadLine()
    lineU = uRead.ReadLine()
    'status.Text = count.ToString
    count += 1
Loop
count = 0

'run through upstream1000 files and input observed / expcted values
into three arrays
Do While count < maskedMultiVals(3)

    lineM = mRead.ReadLine()
    lineU = uRead.ReadLine()
    lineA = aRead.ReadLine()

    'split at blank spaces to get observed frequencies
    Dim splitM() As String = lineM.Split(" ")
    Dim freqM As Decimal = splitM(1)
    Dim splitU() As String = lineU.Split(" ")
    Dim freqU As Decimal = splitU(1)
    Dim splitA() As String = lineA.Split(" ")
    Dim freqA As Decimal = splitA(1)
    arrayMOTIF(count) = splitA(0)
    currMotif = arrayMOTIF(count)

    Dim temp1, temp2, temp3, temp4 As Decimal

    'to complete the random 4/5 roulette wheel data
    If runTYPE = "45" Then
        'ping database for upstream host count because there are not
        preprocessed files
        ' like there are for masked, unmasked, and upstream all
        lineO = getCount(currMotif, chrom, 6)
    
```

```

        'get the expected count for masked
        McurrExp = EXPing(currMotif, chrom, 1)
        'we divide here by 49.45331 due to fact that there are N's
dispersed      ' in the masked data that produces a lower number of total
hits          McurrExp = McurrExp * 49.45331

        'get the expected count for unmasked
        UcurrExp = EXPing(currMotif, chrom, 2)
        UcurrExp = UcurrExp * 50

        'we use the static 1 for chrom bc there's only one table for
upstreamAll   'get the expected count for upstreamAll
              AcurrExp = EXPing(currMotif, 1, 4)
              AcurrExp = AcurrExp * 50

        'get the expected count for upstreamHost
        'because upstreamHost is smaller than the other datasets we
simulate      ' at the full size and therefore do not have to divide by 50
              OccurrExp = getCount(currMotif, chrom, 3)

        'to get our frequency ratio we divide the observed by the
expected     ' and round to 3 decimal points
              arrayM(count) = Math.Round((freqM / McurrExp), 3)
              arrayU(count) = Math.Round((freqU / UcurrExp), 3)
              arrayA(count) = Math.Round((freqA / AcurrExp), 3)
              arrayO(count) = Math.Round((lineO / OccurrExp), 3)

        'to complete the naive approach data
        ElseIf runTYPE = "n" Then
            lineO = getCount(currMotif, chrom, 6)
            arrayM(count) = Math.Round((freqM / expM), 3)
            arrayU(count) = Math.Round((freqU / expU), 3)

            arrayA(count) = Math.Round((freqA / expA), 3)
            arrayO(count) = Math.Round((lineO / expH), 3)
            temp1 = arrayO(count)
            temp1 = temp1 / 1

        End If
        count += 1
    Loop

    count = 0
    Dim nextCount As Integer = 0
    Dim prevCount As Integer = 0
    Dim loopCount As Integer
    Dim motif As String
    Dim growPrevious As Boolean
    Dim mp, up, hp, ap, mz, uz, hz, az, mo As String

```

```

'run through the 3 arrays for frequency bias comparison
Do While count < maskedMultiVals(3)

    'function returns the total count of the motif
    mo = arrayMOTIF(count)
    'these functions return the p-values for each motif
    mp = mpGet(mo, 17, 1)
    up = mpGet(mo, 17, 2)
    hp = mpGet(mo, 17, 3)
    ap = mpGet(mo, 1, 4)
    'these functions return the z-values for each motif
    mz = mzGet(mo, 17, 1)
    uz = mzGet(mo, 17, 2)
    hz = mzGet(mo, 17, 3)
    az = mzGet(mo, 1, 4)
    'output the results to the output file
    freqOut.WriteLine(arrayMOTIF(count).ToString + " " +
arrayA(count).ToString + " " + arrayO(count).ToString + " " +
arrayU(count).ToString + " " + arrayM(count).ToString + _
    " " + mp.ToString + " " + up.ToString + " " + hp.ToString + " " +
ap.ToString + " " + mz.ToString + " " + uz.ToString + " " + hz.ToString
+ " " + az.ToString)

    Loop

    'close file streams
    mRead.Close()
    uRead.Close()
    aRead.Close()
    out.Close()
    freqOut.Close()

    findMotifActually(refSeqLine)
    globalCount += 1

Loop

rRead.Close()

MsgBox("FINISHED!!!!!!")

End Sub

```

R002594117

VITA

Kevin Riehle has a Bachelor of Science degree from the University of Dayton in Biology, 2005 and expects to receive a Master of Computer Science from the University of Dayton, December 2008. Dr. Sudhindra Gadagkar of the University of Dayton is serving as Kevin's thesis advisor.

Kevin has on-going interests in the field of Bioinformatics, and has extensive experience with sequence manipulation and evaluation. Kevin has extensive experience coding in C, C++, VB.NET, and MySQL and has a solid knowledge of Perl and Java.