12-2017

Computational Identification of Noncoding Driver Mutations Based on Impact on RNA Processing

Kevin Zhu

COMPUTATIONAL IDENTIFICATION OF NONCODING DRIVER MUTATIONS

BASED ON IMPACT ON RNA PROCESSING

by

Kevin Wen Zhu, B.A.

APPROVED:

_____
Jeffrey T. Chang, Ph.D.
Advisory Professor

_____
Alemayehu A. Gorfe, Ph.D.

_____
Nicholas E. Navin, Ph.D.

_____
Sanjay S. Shete, Ph.D.

_____
Xiaobing Shi, Ph.D.

APPROVED:

_____
Dean, The University of Texas
MD Anderson Cancer Center UTHealth Graduate School of Biomedical Sciences

COMPUTATIONAL IDENTIFICATION OF NONCODING DRIVER MUTATIONS BASED ON IMPACT ON RNA PROCESSING

A

THESIS

Presented to the Faculty of

The University of Texas

MD Anderson Cancer Center UTHealth

Graduate School of Biomedical Sciences

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE

by

Kevin Wen Zhu, B.A.

Houston, TX

August 2017

Dedication

This thesis is dedicated to my parents, without whom I would not be where I am today.

Acknowledgements

First and foremost, I would like to acknowledge my advisor, Dr. Jeffrey Chang. Without his knowledge, patience, and support, I would never have been able to accomplish what I did. Additionally, I would like to thank and acknowledge the other members of my committee for time and support: Dr. Alemayehu Gorfe, Dr. Nicholas Navin, Dr. Sanjay Shete, and Dr. Xiaobing Shi. I would also like to thank Dr. Xiaodong Cheng, who kindly agreed to substitute for a member of my committee who was unable to attend my defense.

I would also like to acknowledge the other members of the lab, Dr. Sara Prijic and Dr. Weina Zhao for their patience during the lab meetings in which I presented and their advice, recommendations, and encouragement.

My greatest appreciation goes out to my parents for their support and understanding, both when raising me and during my time as a graduate student. I also would like to thank my sister and her husband, as well as my cousin and her family for their support.

Finally, I would like to acknowledge Dr. Bill Mattox, Mrs. Brenda Gaughan, and the other members of the GSBS academic office for their patience and help in guiding me through my graduate school career.

COMPUTATIONAL IDENTIFICATION OF NONCODING DRIVER MUTATIONS BASED ON IMPACT ON RNA PROCESSING

Kevin Wen Zhu, B.A.

Advisory Professor: Jeffrey T. Chang, Ph.D.

Despite the prevalence of mutations in the noncoding regions of the DNA, their effects on cancer development remain largely uninvestigated. This is especially evident when compared to coding mutations, which have been relatively well-studied and, in certain cases, been identified as driver mutations for cancer. Recent studies, however, have identified noncoding mutations that frequently appear in certain types of cancer, which may be evidence that those mutations are important to cancer development. Nonetheless, the role of noncoding mutations in cancer remains unclear. A potential vector for understanding this mechanism is through observing the relation between noncoding mutations and functional RNA motifs. The goals for this study, therefore, were to identify RNA motifs that were significantly associated with the presence of somatic, noncoding mutations and to predict the functional impact of noncoding variants. The analysis was conducted on mutations detected in whole genome sequencing profiles of breast cancer samples obtained from the TCGA database. I derived the significance of the number of noncoding mutations affecting a particular motif as well as the enrichment of noncoding mutations on each motif. I also created linear models to identify the motifs with mutations that had the greatest impact on cancer-related pathways. I found that a number of motifs are affected by significantly less mutations than we would expect at random. Additionally, I found that functional RNA motifs related to splicing are often

significant in the linear models, suggesting that they play a role the relation between noncoding mutations and cancer. These findings will help improve understanding of the effects of noncoding mutations on RNA processing in the context of breast cancer.

TABLE OF CONTENTS

List of Figures

List of Tables

INTRODUCTION

1. Driver Mutations

Random mutations that occur during DNA replication are the leading cause of known cancer incidences (Tomasetti and Vogelstein, 2015). However, only a portion of these mutations promote tumorigenesis. Driver mutations provide a selective advantage that allows a cell or cell population to grow and avoid cell death or senescence. Other mutations, known as passenger mutations, do not provide such an advantage, but may accompany the driver mutations during clonal expansion because of their presence in the same genome. Passenger mutations may have arisen before or after the clonal cells have developed into cancer (Wood, et al., 2007).

2. Coding and Noncoding Mutations

Until recently, cancer driver mutations have been found only among mutations in the coding regions of the genome, or those regions that are eventually transcribed into proteins. This is because it is relatively easy to predict the consequences or impact of mutations on the encoded protein, which can be either gain or loss of protein function due to changes in expression, folding, trafficking, or posttranslational modification. For this reason, the majority of the sequence data in cancer sequencing database only contain exon sequences, leaving possible driver variants in the noncoding sequences, which are genomic sequences that do not encode protein sequences, undetected.

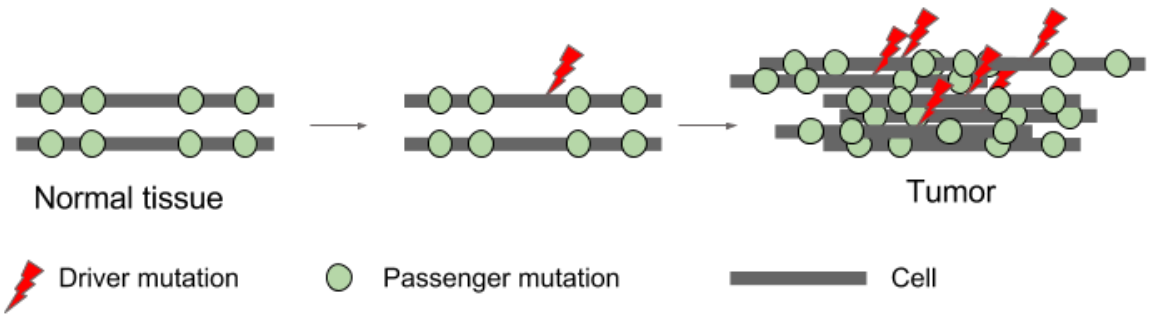

Figure 1. How a driver mutation can cause normal tissue to develop into cancer. Pre-existing passenger mutations do not contribute to the development of the cancer. Once a driver mutation affects a cell, the cell gains a selective advantage that allows it to grow and cause the tissue to become tumorous.

Typically, driver coding mutations have been identified based on their frequency in cancer samples or by their functional impact. In the former, genes that are mutated in a significantly greater number of cancer samples when compared to a background rate are regarded as candidate driver genes. Somatic coding mutations on these genes are then identified as candidate drivers based on the consistency and predictability of their effects on the function of the protein encoded by the gene. Functional impact is then used to differentiate between driver and passenger mutations by identifying the mutations that generally have a stronger impact on protein function as it relates to cell growth and survival (Pon and Marra, 2015).

However, it is estimated that coding regions account for only 2% of the human genome (Elgar and Vavouri, 2008; Lee et al., 2010), which means that noncoding regions that make up the other 98% have largely been ignored when identifying possible cancer driver mutations. As shown in Figure 2, there are a proportional number of noncoding variants in the breast cancer samples when whole genome sequences are considered. This means that a large number of mutations have yet to be studied. There are several reasons for the lack of research into noncoding mutations in cancer, among which the biggest issue is the sample size. As noted, the sheer number of variants in the noncoding regions far exceeds that of those found in the coding regions, which can lead to difficulty in identifying recurrent mutations with statistical significance. Current solutions have focused on transcriptional regulatory regions, but this ignores the fact that gene products are

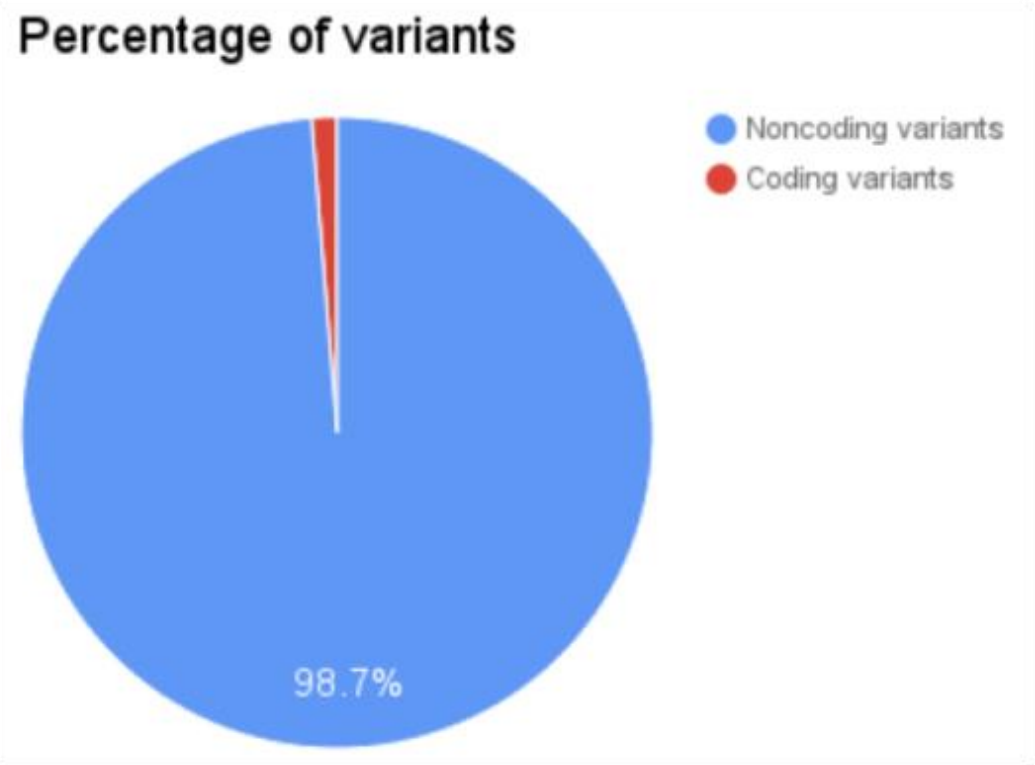

Figure 2. Average percentage of coding and noncoding variants across 28 TCGA breast cancer samples analyzed in this study.

processed in other ways that affect function. Therefore, the effects of noncoding mutations are more difficult to discern compared to those of coding mutations (Elgar and Vavouri, 2008).

3. History

Because of our limited understanding of noncoding variants and their effects, it is difficult to discern if any such variants could be considered a driver mutation and the large number of noncoding variants makes it difficult to even identify potential candidates. Nonetheless, recent efforts have been made to facilitate finding important noncoding mutations by restricting the scope of the population, typically by searching for mutations recurrent across multiple samples. For example, one important study identified potential driver mutations in the promoter region of *TERT* by analyzing a number of melanoma samples for recurrent mutations and finding two that were collectively present in 70% of melanoma cases studied (Huang et al., 2013). Since then, other studies have further restricted the number of noncoding mutations by using techniques that rely on identifying recurrent mutations based on genomic regions. For example, hotspot analysis identifies short regions of the genome (e.g., 50 bps) in which more mutations appear relative to nearby regions. Another method is the regional mutation recurrence, which looks for sets of regions with similar properties, such as nucleotide makeup or other sequence features, and identifies the sets that carry more mutation compared to other sets (Weinhold, et al., 2014). In one study, for example, researchers identified a notable enrichment of mutations in the regulatory elements of *ESR1* in 7% of ESR1-positive breast

cancers, compared to just 1% of somatic copy number alterations in the same cancer type (Bailey et al., 2016). Another common approach is to search areas where important mutations are known to occur, such as the promoter regions as identified in the *TERT* study. *FOXA1*, along with a number of other genes, has been identified in breast cancer as a potential driver gene based on recurrent mutations present in its promoter region (Rheinbay et al., 2017).

Other methods that have been used to facilitate looking through noncoding mutation is by attempting to discern which mutations have more significant effects based on gene expression (Fredriksson, et al., 2014). This method involves evaluating genes for associations between RNA levels and noncoding somatic mutations. While the mutations were originally identified via location-based methods, as mentioned previously, the researchers in the FOXA1 study observed that the mutations in the *FOXA1* promoter region impacted gene expression and caused an increase in the amount of FOXA1 protein, which would then result in increased sensitivity to estrogen in affected cells (Rheinbay et al., 2017). This allowed them to not only further confirm their observation, but also develop another possible method for identifying candidate noncoding driver mutations.

4. <u>Functional RNA Motifs</u>

While there has been progress in identifying the mechanisms connecting noncoding mutations and cancer, current analyses has been limited due to largely focusing on transcriptional regulatory regions. Since gene products are processed in other ways that can affect function, it may be possible to improve identification of

noncoding driver mutations by looking at regions that affect other aspects of non-transcriptional regulation. One potential vector for understanding the general relation between noncoding variants and cancer is through exploring their connection with RNA processing. As RNA matures, it goes through a sequence of steps that impact the regulation of gene expression, alternative splicing, transcriptional termination, and others. The sequences of elements predictive of sites of processing have been represented as motifs and can be predicted from the sequence (Chang et al., 2013). So far, however, there has been little research into whether RNA motifs can be used to identify regions of the genome enriched for noncoding mutations. Therefore, in order to examine the contributions of noncoding mutations to tumorigenesis, the goal of this thesis was to develop computational approaches to identifying driver mutations in the noncoding regions of the genome that impact cancer development based on their effect on functional RNA motifs.

5. Breast Cancer

Breast cancer stands out as one of the most prevalent cancers among women in the United States and the second-leading cause of cancer-related death among women after lung and bronchus cancer (American Cancer Society, 2016). Because of this, there is a relatively large amount of available sequencing data, including whole genome sequences, for breast cancer (Weinstein et al., 2013). The large dataset is very important for statistical evaluation in computational methods. Moreover, while mutations in the noncoding regions of particular genes have been identified in breast cancer, melanoma, bladder cancer, glioblastomas, and other

cancers, there are still a large number of noncoding mutations that have not been analyzed due to the focus on transcriptional regulatory regions in previous studies (Huang et al., 2013; Horn et al., 2013; Fredriksson et al., 2014; Weinhold et al., 2014). Thus, noncoding mutations remain poorly explored in cancer research. Although there is still much that needs to be done before the relation between noncoding mutations and cancer is fully understood, focusing on breast cancer may yield important insights on possible driver mutations in the noncoding regions that impact cancer development, which would benefit patients and society.

HYPOTHESIS AND SPECIFIC AIMS

I hypothesize that functional noncoding mutations are enriched in genomic sequences that impact RNA processing.

To test the hypothesis, I identified two specific aims (Figure 3).

Aim 1: To develop computational methods for identifying RNA motifs that are significantly associated with the presence of somatic, noncoding mutations.

Approach: I first predicted RNA motifs, then performed a statistical analyses in order to identify those that were significantly associated with noncoding mutations. For the statistics, I compared the data against similar, but random locations in the genome to identify if there was a significant difference in the number of mutations affecting a particular motif.

Results: I discovered that some of the motifs have a significant association with noncoding mutations. Each motif that showed significance displayed negative enrichment of mutations affecting the motifs relative to the randomly selected locations, implying that the number of mutations affecting those motifs was less than would be expected at random.

Aim 2: To predict functional impact of noncoding variants.

Approach: I looked for potential associations between RNA motifs affected by noncoding mutations and known cancer pathways. I estimated the activation of the pathways using gene expression information. I then applied regression analysis to identify RNA motifs predictive of functional activity.

Results: This analysis revealed that certain motifs consistently have a significant association with the cell cycle pathway. Expanding the analysis to other pathways

revealed that splicing-related motifs were often significant in the pathways that were most impacted. This suggests that splicing-related motifs have a significant impact in the relation between noncoding motifs and breast cancer.

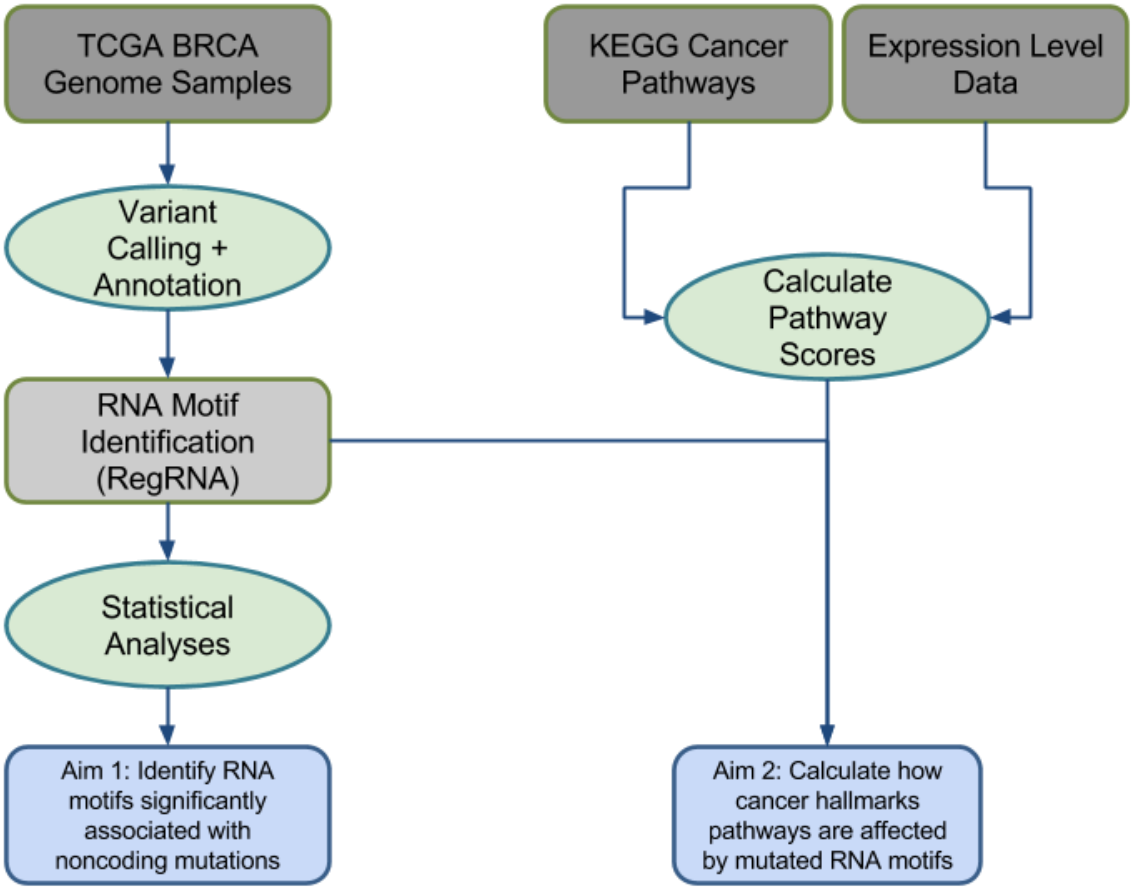

Figure 3. Overall pipeline to accomplish specific aims

METHODS

1. Data Processing and Variant Identification

I obtained whole genome sequenced (WGS) breast cancer (BRCA) samples from The Cancer Genome Atlas (TCGA) (Weinstein et al., 2013). The samples I used for the study were primarily those with the largest file sizes, which suggested that they would have greater read depth. Each sample consists of a patient's tumor sample as well as a paired normal sample, which allows us to filter out germline and SNPs during variant calling. Of the 1084 patients with breast cancer samples available from the TCGA, 118 had WGS analysis done on them. BAM files were downloaded from the Cancer Genomics Hub (CGHub) or the Genomic Data Commons (GDC). Samples originated either from the Washington University Genome Sequencing Center (WUGSC) or Raju Kucherlapati's lab at the Harvard Medical School (HMS-RK). Because the preprocessing differs between the sources, I first preprocessed the files again to ensure consistency. To do this, I used an in-house program known as BETSY, which was in our lab (Chen and Chang, 2017). I realigned the sequences using BWA-MEM (Li, 2013), marked duplicates, added read groups, realigned insertions and deletions, and sorted by coordinates (Desprito, et al., 2011). The list of tumor samples and their corresponding normal samples are presented in Table 1.

After I processed both the tumor and normal samples for a patient, I used the variant caller MuTect to identify the possible mutations (Cibulskis et al., 2013). MuTect takes as input the paired patient samples as well as a reference genome,

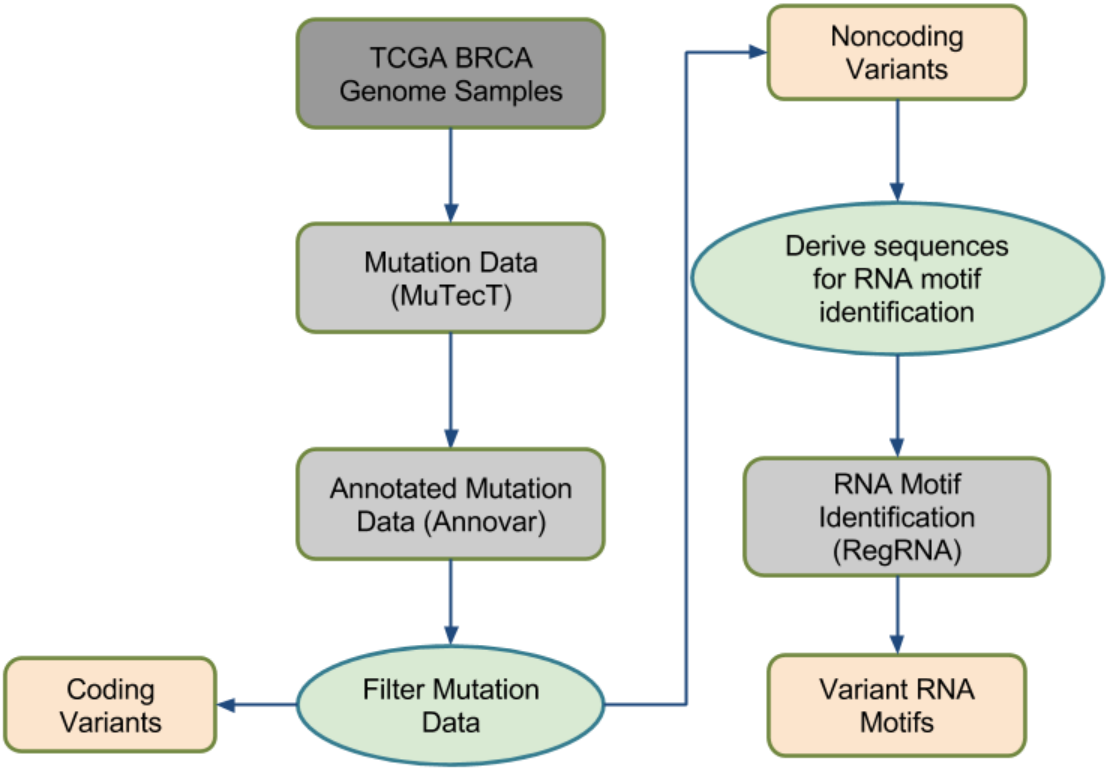

Figure 4. Pipeline for data processing

TCGA ID (Tumor File)	TCGA ID (Normal File)	Center
TCGA-B6-A0X4-01A-11D-A106-02	TCGA-B6-A0X4-10A-01D-A106-02	HMS-RK
TCGA-B6-A0RE-01A-11D-A060-02	TCGA-B6-A0RE-10A-01D-A128-09	HMS-RK
TCGA-E2-A15K-06A-11D-A12Q-09	TCGA-E2-A15K-10A-01D-A12Q-09	WUGSC
TCGA-E2-A109-01A-11D-A10M-09	TCGA-E2-A109-10A-01D-A10M-09	WUGSC
TCGA-A7-A0CE-01A-11D-A12L-09	TCGA-A7-A0CE-11A-21D-A12L-09	WUGSC
TCGA-A2-A0D2-01A-21D-A128-09	TCGA-A2-A0D2-10A-01D-A128-09	WUGSC
TCGA-EW-A3U0-01A-11D-A228-09	TCGA-EW-A3U0-10A-01D-A22A-09	WUGSC
TCGA-E2-A1LG-01A-21D-A14K-09	TCGA-E2-A1LG-11A-42D-A14K-09	WUGSC
TCGA-A7-A13D-01A-13D-A12Q-09	TCGA-A7-A13D-10A-02D-A12Q-09	WUGSC
TCGA-A7-A26G-01A-21D-A167-09	TCGA-A7-A26G-10A-01D-A314-09	WUGSC
TCGA-D8-A27H-01A-11D-A314-09	TCGA-D8-A27H-10A-01D-A17G-09	WUGSC
TCGA-A2-A259-01A-11D-A314-09	TCGA-A2-A259-10A-01D-A17G-09	WUGSC
TCGA-AN-A0AT-01A-11D-A045-09	TCGA-AN-A0AT-10A-01D-A047-09	WUGSC
TCGA-EW-A1P8-01A-11D-A142-09	TCGA-EW-A1P8-10A-01D-A314-09	WUGSC
TCGA-BH-A18R-01A-11D-A19H-09	TCGA-BH-A18R-11A-42D-A19H-09	WUGSC
TCGA-BH-A0DT-01A-21D-A12B-09	TCGA-BH-A0DT-11A-12D-A12B-09	WUGSC
TCGA-AN-A0G0-01A-11D-A045-09	TCGA-AN-A0G0-10A-01D-A047-09	WUGSC
TCGA-E2-A152-01A-11D-A19H-09	TCGA-E2-A152-10A-01D-A19H-09	WUGSC
TCGA-A2-A3KC-01A-11D-A20S-09	TCGA-A2-A3KC-10A-01D-A20S-09	WUGSC
TCGA-A8-A094-01A-11D-A19H-09	TCGA-A8-A094-10A-01D-A19H-09	WUGSC
TCGA-AO-A03L-01A-41D-A19H-09	TCGA-AO-A03L-10A-01D-A19H-09	WUGSC
TCGA-A8-A075-01A-11D-A19H-09	TCGA-A8-A075-10B-01D-A19H-09	WUGSC
TCGA-A2-A04Q-01A-21D-A128-09	TCGA-A2-A04Q-10A-01D-A128-09	WUGSC
TCGA-B6-A0I6-01A-11D-A128-09	TCGA-B6-A0I6-10A-01D-A128-09	WUGSC
TCGA-AO-A0JM-01A-21D-A19H-09	TCGA-AO-A0JM-10A-01D-A19H-09	WUGSC
TCGA-AO-A0J2-01A-11D-A19H-09	TCGA-AO-A0J2-10A-01D-A19H-09	WUGSC
TCGA-B6-A0IJ-01A-11D-A128-09	TCGA-B6-A0IJ-10A-01D-A128-09	WUGSC
TCGA-B6-A0RT-01A-21D-A128-09	TCGA-B6-A0RT-10A-01D-A128-09	WUGSC

Table 1. List of tumor sample IDs, matched normal sample ID, and the originating lab

thus allowing it to filter out germline mutations. Additionally, MuTect will accept as input files containing data from the COSMIC database and the dbsnp database, which are used as additional references for MuTect's calculations. As output, MuTect presents its results as a tab-delimited text file, which includes columns for various statistics, such as read depths for both the tumor and normal sample and the allele frequencies for reference and variant alleles in both samples. Furthermore, the caller can suggest whether a particular variant may be an actual variant or a false positive, while providing justification for its decision.

2. Variant Annotation and Filtering

Once I had identified the potential variants in a sample via MuTect, I used ANNOVAR to annotate them (Wang, et al., 2010). These annotations identify aspects of the mutation, such as where a variant is located on a gene (e.g., intron, exon, intergenic) and which gene the variant affects, or the closest genes if the variant is intergenic.

My criteria for selecting the variants used for the study are based on the statistical information derived from MuTect's results. Specifically, I filtered out any variants that the caller has marked as unlikely to be a real mutation. I also filtered out any variants that fell in the exonic regions of genes as identified by ANNOVAR, since I wanted to primarily focus on mutations in the noncoding regions. In order to further ensure the accuracy of the detected variants, I also filtered out those that have a read depth of 30 or less. Finally, I removed variants that were not within 25,000 base pairs of an end of a gene. This filtering process was done via Python scripts.

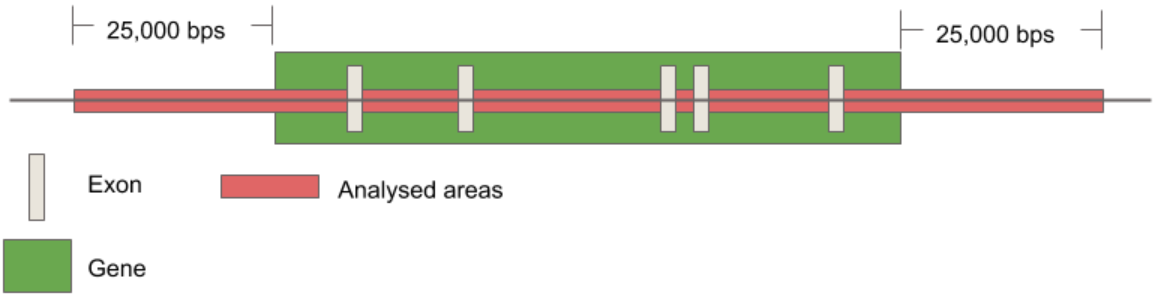

Figure 5. Areas of the DNA considered for analysis of noncoding variants.

To further reduce the number of variants and to begin identifying any potential associations, I restricted the list of genes based on the number of samples in which the gene is mutated. I extracted the list of genes from the ANNOVAR results for each sample, then constructed a table indicating whether a gene carried a mutation in a particular sample. After totaling up the number of samples each gene is present in, I considered only genes that have variants in at least 8 different samples. The set of noncoding variants was then reduced to only those that affected one of the genes in the list, or was near one of the genes if the variant was intergenic. The number of variants at each step can be found in the Results section.

3. Functional RNA Motif Identification

Once I had finalized the set of noncoding variants, I began identifying the functional RNA motifs that were affected by the variants. This was done using the integrated web server for RegRNA 2.0 (Chang et al., 2013), which takes as input a sequence of nucleotides of either DNA or RNA and outputs information on any functional RNA motifs that were identified in the sequence. Due to limitations with the RegRNA program, I could only analyze sequences of up to 10,000 base pairs at a time. For sequences of that length, the program took up to a minute to analyze. Therefore, I attempted to select the sequences such that as many variants as possible would be covered with every scan. I used a Python script to evaluate the mutations and identify sequences of 10,000 base pairs that covered the most variants. In order to account for RNA motif sequences that begin before the mutation's position, the script set the start of the sequence for analysis at 200 base

pairs before the mutation, which is a large enough buffer for the majority of possible motifs. I then used the Python library *mechanize*, which allows a user to browse web pages via Python, to input the sequence to RegRNA and obtain the output, which were then stored for later analysis. In order to avoid overloading the RegRNA server, I added a 45 second delay between requests.

RegRNA outputs are formatted as tab-delimited text-files with columns indicating the motif type, the motif name, the range of bases, the length of the motif sequence, and the sequence itself. Using this information, I determined the RNA motifs affected by a particular variant by identifying the 10,000 base pair sequence that contained the variant, then analyzing the matching file to get the data. I used a Python script to scan each line of the file and identify the ones that contained the variant's position in the listed range of bases.

4. <u>Identifying RNA motifs significantly affected by noncoding mutations</u>

To identify RNA motifs enriched for mutations, I compared the motifs affected by mutations against a background. The background data for this comparison needed to be free from bias. I first attempted to use the RNA motif found in all of the analyzed sequences as the background data. This, however, was biased, since the analyzed sequences were based on the set of noncoding variants. Therefore, I decided to instead randomly select segments of the genome that were within the same bounds as those used to filter the variant sets. While this seemed to work better as the background data set, I realized that it could be further improved by selecting random locations on the noncoding segments. Specifically, I obtained the

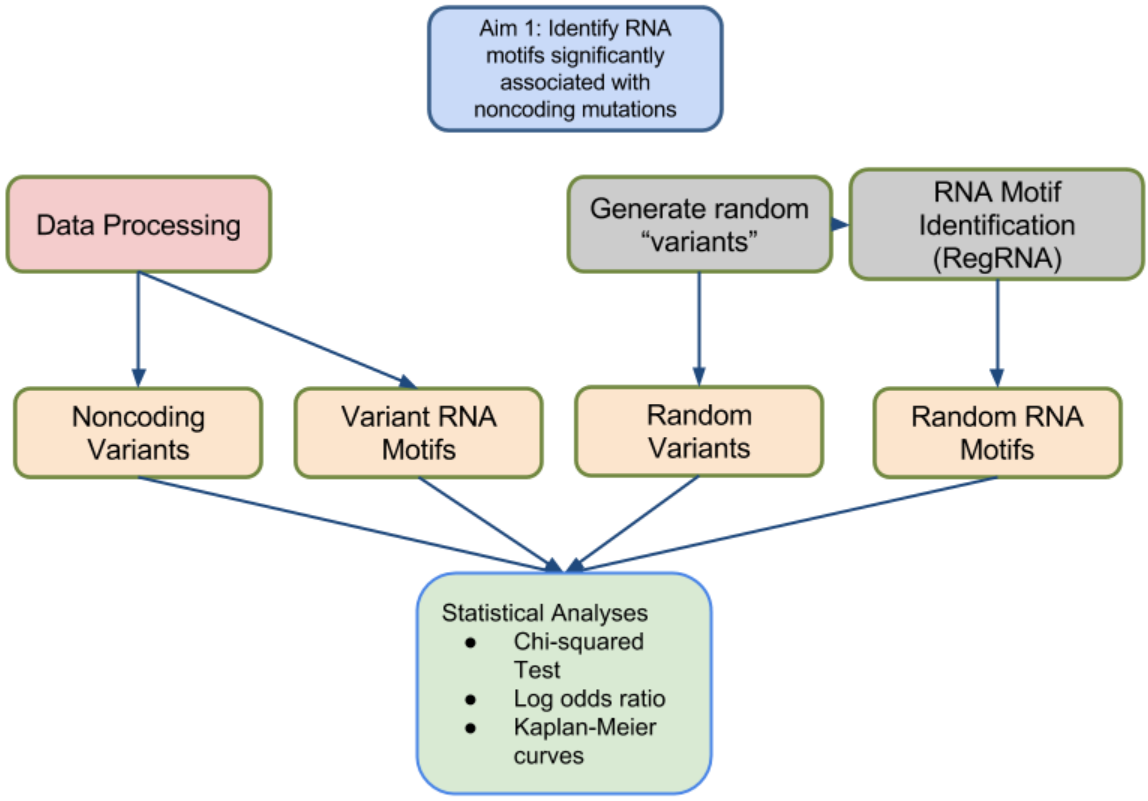

Figure 6. Pipeline for first specific aim

genomic start and end positions for each gene and the positions of the gene's exons from the UCSC genome browser and randomly selected points such that the points were within 25,000 base pairs of the end of a gene and did not lie in any gene's exonic regions. These points were defined as noncoding variants for the random set. I then analyzed the sequences that cover these random points, therefore providing us a background data set similar to our actual data.

In order to check for bias in the random selection, I independently generated two different random sets of points using the criteria described previously. The number of points in each set was approximately the same as the number of actual noncoding variants. After I analyzed the points to find the motifs affected, I used a 2x2 contingency table for each motif to calculate the chi-squared significance between the random sets of points. The columns of the tables totaled the number of bases that were considered part of the first random set or the second random set, while the rows totaled the number of bases that were part of the particular motif sequence. I applied a Bonferroni correction and found that none of the motifs showed any significant difference between the random sets. This indicated that the selection process could generate background sets with similar properties.

To score the RNA motifs, I used the same technique. I constructed contingency tables for each motif containing the counts of variants compared to one of the random sets. I again applied a Bonferroni correction and evaluated which motifs were significant. If a particular RNA motif was shown to be significant based on the p-value of its chi-square score, then this indicated that the number of mutations that affected that motif was significantly different from what would be

expected at random. I also calculated the log-odds ratio for each motif, which gave further information on the enrichment of variants affecting motifs in the variant set relative to the background data. If the ratio was positive for a particular motif, this indicated that there were more mutations affecting the motif in the breast cancer samples than would be expected at random; conversely, if the ratio was negative, then the motif had less mutations affecting it. I did these analyses for both of the random sets that I had generated, which allowed me to check if the results were consistent.

5. <u>Regression analysis of motifs</u>

Using the list of noncoding mutations and the affected RNA motifs, I determined their functional significance by analyzing the pathway scores. As a positive control, I used the coding variants in the analysis since I would expect those variants to have a significant effect on cancer-related pathways. I obtained the information for the KEGG cancer pathways from SigDB (Kanehisa et al., 2017; Kanehisa et al., 2016; Kanehisa et al., 2000), which included the pathway names and the genes that made up the pathways. In order to calculate a score for the pathways, I also downloaded the file containing breast cancer gene expression information from the Broad Firehose Pipeline (Broad Institute TCGA Genome Data Analysis Center, 2016). This file consists of a table of genes that were analyzed for their expression levels against TCGA breast cancer samples. Using an in-house program, I calculated a score for each pathway based on the genes in the pathway

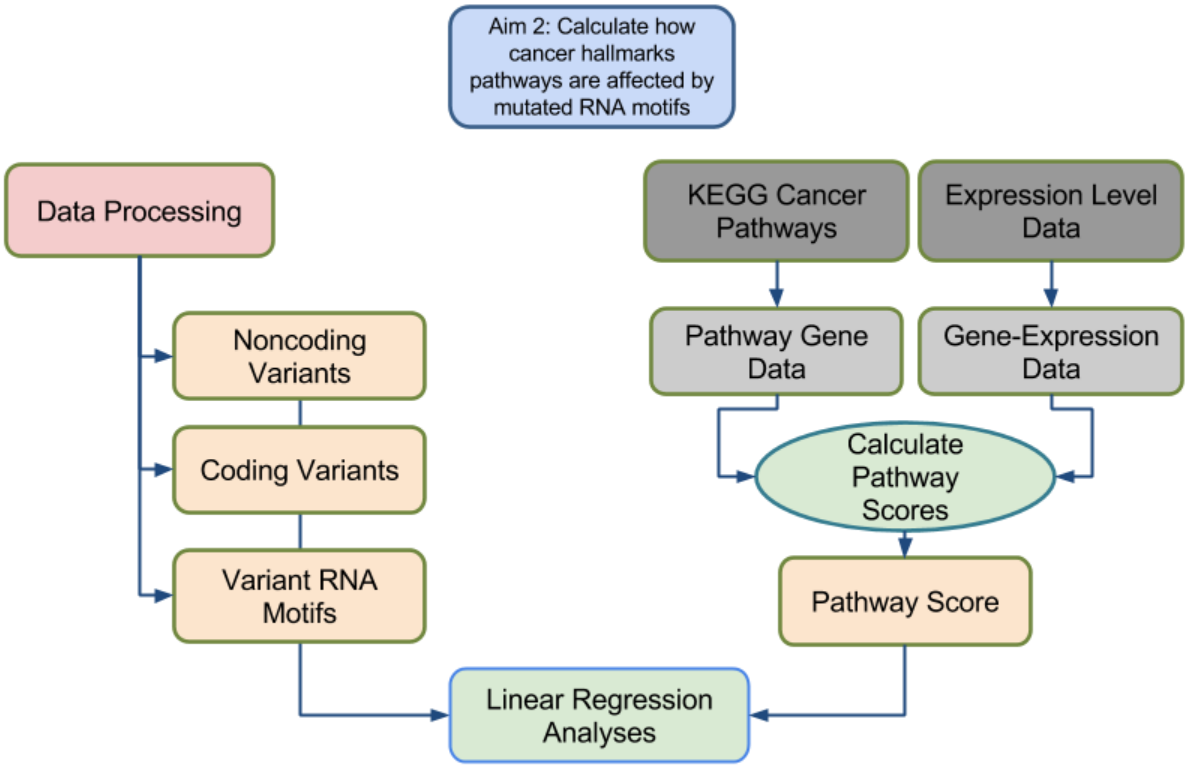

Figure 7. Pipeline for second aim

and their expression levels (Chang et al., 2011). I then constructed a simple linear model in R using the pathway score as the response variable, while the explanatory variables were selected based on the desired model. In order to ascertain the effectiveness of the model, I first focused on the cell cycle pathway, which is known to be a highly important pathway in cancer (Hartwell and Kastan, 1994). The first model I constructed used only the coding mutations as the explanatory variables in order to test whether they could have a significant effect on the cell cycle's pathway score, validating my approach. As shown in Table 2, the p-value indicates that the coding mutations do indeed have a significant effect in determining the pathway score. Once I had verified that the coding mutations contributed significantly in the linear model, I constructed models to see if any of the motifs affected by noncoding mutations could also have a significant effect on the score. To do this, I counted the number of mutations that affected each motif in each sample. To account for the fact that the number of mutations affecting a motif varied directly with the number of mutations in the sample, I divided the number of mutations that affected a particular motif by the total number of mutations affecting any motif in the sample, thus calculating the proportion of all mutations affecting a specific motif. Using these proportions as the explanatory variables, I constructed three models: the first

Variable	Coefficient	p-value
Coding	116.63	0.00379***

Table 2. Coefficient for a linear model with the coding mutations as explanatory variables. ***P < 0.01

consisting of all noncoding motifs, the second consisting of all noncoding motifs and the coding mutations, and the last consisting only of those motifs found to be significant in aim 1 plus the coding mutations.

Once I had evaluated the effects of the RNA motifs on the cell cycle pathway score, I expanded my analysis to the other pathways in KEGG's set of pathway. Since the p-value for the model using only the significant RNA motifs and the coding mutations was the most significant, I applied that model to the other pathways. After calculating the coefficients, I used an in-house program to calculate the false discovery rate (FDR) adjusted p-values for each of the pathways. I then pulled out the five pathways with the most significant FDR adjusted p-value, which are listed in Table 6. For each of these pathways, I identified the RNA motifs that were significant for $p < 0.05$. This allowed me to identify if there were any patterns in which biological processes defined by motifs were most commonly differentially affected by noncoding mutations.

RESULTS

1. Data Processed

I downloaded a total of 28 paired BRCA WGS files from the TCGA. Of the samples, 2 came from the HMS-RK, while 26 came from the WUGSC. All of the files were reprocessed as described in the Methods, then analyzed using MuTect. After filtering out all variants that were rejected based on MuTect's own criteria, there was a total of 303,811 unique variants detected across all samples. After I had annotated the variants with ANNOVAR, I separated them into the coding and noncoding variants. This gave a total of 2,872 coding variants and 300,939 noncoding variants. Of the remaining noncoding variants, I filtered out those that did not satisfy the criteria stated in the Methods section, leaving a total of 164,289 unique variants. Finally, when I considered only the genes that were affected by noncoding variants in at least 8 samples, I obtained 1,981 unique genes, thereby providing a total of 43,501 noncoding variants in the final set. 1,352 of these variants were present in more than one sample, with 235 of them present in three or more.

From the variants across all of the samples, I generated 17,507 sequences of 10,000 base pairs. Each sequence was passed through the RegRNA 2.0 web server and analyzed for possible functional RNA motifs. Additionally, two independent sets of randomly-selected points were generated as the background data for comparison with the set of noncoding variants. The first set contained 7,163 points, while the second consisted of 7,154 points. An additional 5,171 sequences

Abbreviation	Full Motif Name	Abbreviation	Full Motif Name
ncRNA	ncRNA hybridization region	ESE	Exon splicing enhancer
longStems	Long stems	ESS	Exon splicing silencer
funcRNA	Functional RNA motifs	UTR	Untranslated region
miRNA	microRNA target sites	ERPIN	Cis-regulatory elements of ERPIN
Transcript	Transcriptional RNA motifs	Splicing	Human splicing sites
RIT	Rho-independent terminator	ARE	AU-rich elements
PAS	Polyadenylation Sites	RBS	Ribosome binding sites
ISE	Intron splicing enhancers	C2U	C-to-U RNA editing sites
ORF	Open reading frames	ISS	Intron splicing silencer

Table 3. Identified functional RNA motifs and name abbreviations.

were analyzed via RegRNA to cover the random points. Table 1 lists the types of motifs that were detected in the breast cancer samples and the random sets of points, as well as the number of variants in each sample or random set that affected the particular motif.

2. Significant RNA Motifs

Of the 19 functional RNA motifs that RegRNA can identify, 16 were affected by at least one variant present in both the cancer samples and the randomly generated points. Out of the 16 motifs, 8 were significantly differentially affected by noncoding variants compared to the first random set. These motifs include transcriptional regulatory motifs, exon splicing silencers, long stems, intron splicing enhancers, microRNA target sites, functional RNA sequences, ncRNA hybridization regions, and open reading frames. The same motifs were also significant when compared to the second random set, though with the addition of exon splicing enhancers, showing that the results were consistent. Additionally, when I compared the log ratios between the variants and both random sets, I found that every significant motif had a negative ratio, as shown in figure 8 for the first random set. Figure 9 shows that the results are consistent for the second random set. What this indicates is that in the cancer samples, those motifs are affected by fewer noncoding variants than we would expect if the motifs had been affected at random. This implies that the motifs may have a critical effect on the survival of cancer cells or perhaps even all cells, including normal cells.

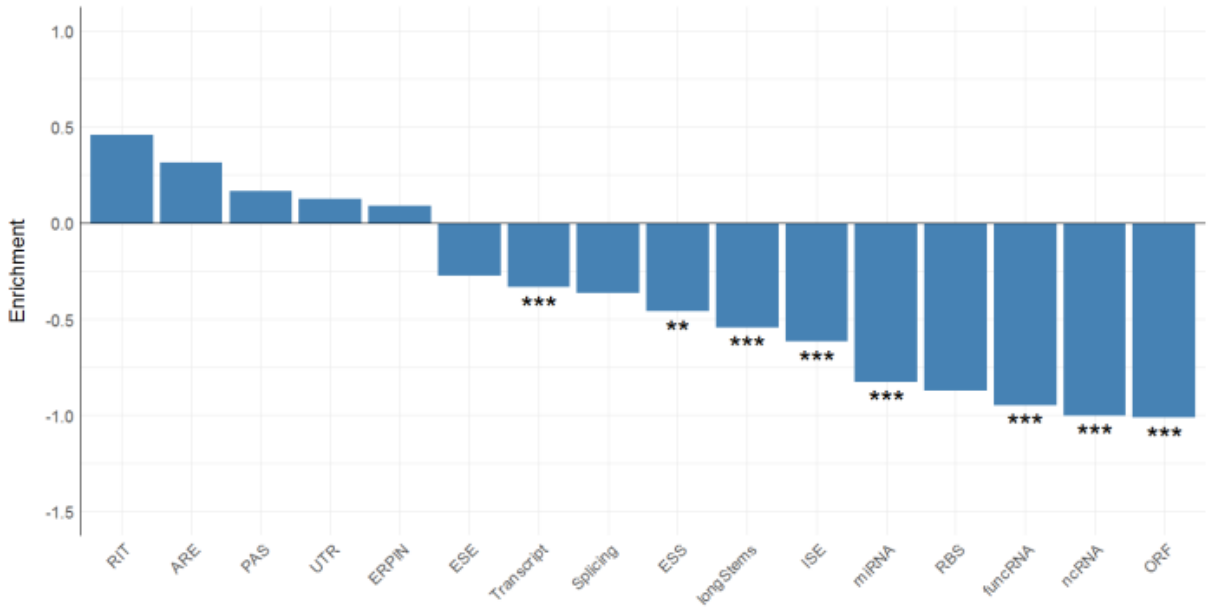

Figure 8. Log-odds ratio between the variants set and the first random set with chi-square significance. ***P < 0.01; **P < 0.05

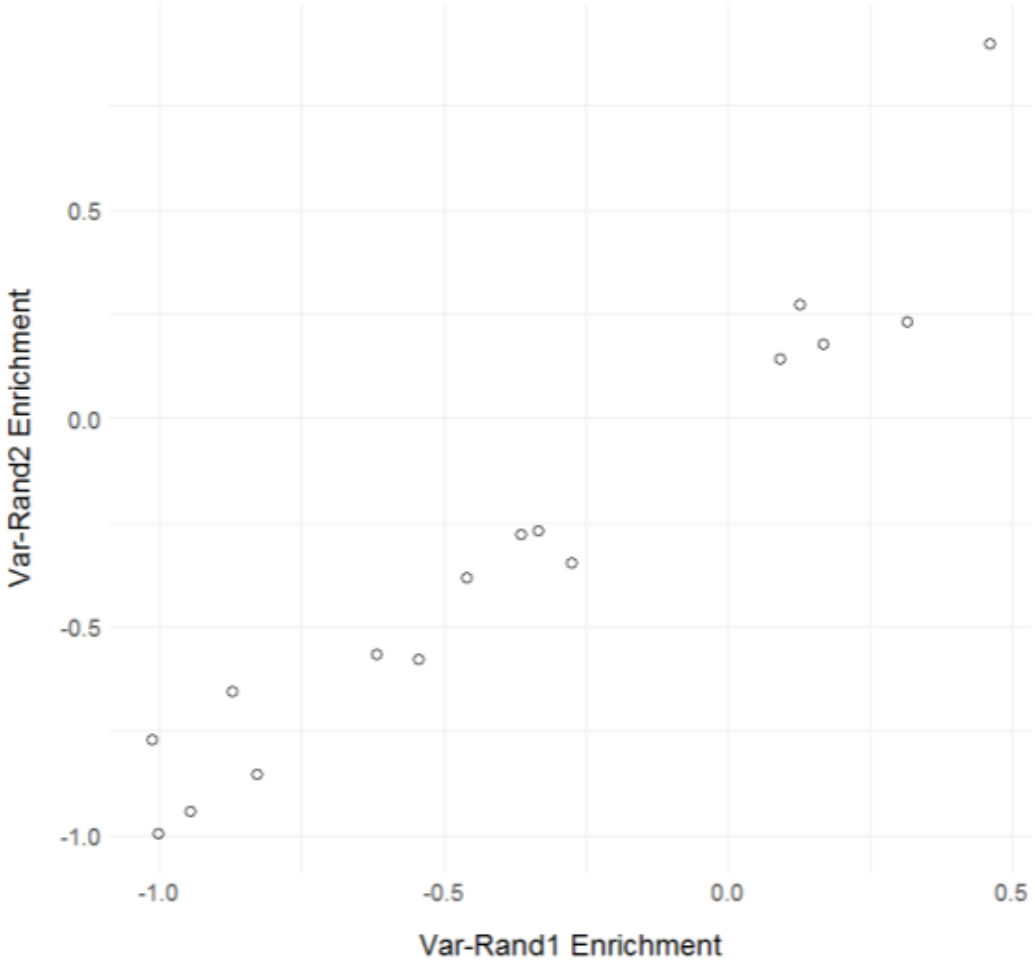

Figure 9. Comparison of log-odds ratios between variants set and first random set and between variants set and second random set.

It would be ideal to find out if the motifs bearing possible cancer-related mutations might affect cancer prognosis by correlating their occurrence with the survival rates of the patients. However, partially due to the low number of cancer samples I have analyzed so far, it is difficult to construct a Kaplan-Meier curve that displays any significance in survival.

3. Significant RNA motifs within a pathway remain constant

The coefficients and p-values for the linear model using all of the RNA motifs affected by noncoding variants as explanatory variables are presented in Table 3. Although the p-values for many of the RNA motifs are fairly weak, there are nonetheless two motifs - the transcriptional regulatory motif and intron splicing enhancer motif - that appear to have a significant effect on the cell cycle pathway score, with $p < 0.05$. Notably, this also holds true for the model that includes all RNA motifs plus the coding mutations and the model that includes only significant motifs plus the coding mutations, which are presented in Table 4 and Table 5, respectively. This suggests that these provide functionality unrelated to the mutations found in the coding regions.

4. Splicing-related RNA motifs appear significant in cancer-related pathways

After applying the false discovery rate adjustment, I found that most of these pathways do not have a significant p-value, with the lowest, the cell cycle pathway,

Variable	Coefficient	p-value	Variable	Coefficient	p-value
ESE	-12.505	0.4587	**ncRNA**	-18.041	0.1132
ESS	-12.215	0.4553	**longStems**	-5.717	0.2068
UTR	-12.384	0.1321	**funcRNA**	22.893	0.0747*
ERPIN	3.292	0.7222	**miRNA**	-4.850	0.7024
Splicing	13.840	0.5727	**Transcript**	7.793	0.0555*
ARE	23.119	0.3135	**RIT**	6.999	0.5946
RBS	8.601	0.9421	**PAS**	-9.777	0.2195
C2U	-722.505	0.2838	**ISE**	-20.407	0.0203**
ISS	157.523	0.5929	**ORF**	1.738	0.6961

Table 4. Coefficient for a linear model with all noncoding motif proportions as explanatory variables. *P < 0.1; **P < 0.05

Variable	Coefficient	p-value	Variable	Coefficient	p-value
ncRNA	-15.495	0.1792	**ESS**	-9.508	0.5642
longStems	-6.325	0.1717	**UTR**	-12.380	0.1343
funcRNA	22.468	0.0818*	**ERPIN**	-5.606	0.6620
miRNA	-7.388	0.5699	**Splicing**	11.275	0.6466
Transcript	9.693	0.0393**	**ARE**	9.667	0.7110
RIT	3.501	0.7956	**RBS**	-8.574	0.6428
PAS	-8.429	0.2930	**C2U**	-736.873	0.2762
ISE	-24.157	0.0165**	**ISS**	87.914	0.7701
ORF	1.351	0.7617	**Coding**	80.586	0.3263
ESE	-18.363	0.3114			

Table 5. Coefficient for a linear model with all noncoding motif proportions and coding mutations as explanatory variables. *P < 0.1; **P < 0.05

Variable	Coefficient	p-value	Variable	Coefficient	p-value
ncRNA	-14.3670	0.1036	**ISE**	-14.0437	0.0220**
longStems	-0.6398	0.7900	**ORF**	-1.5679	0.5791
funcRNA	11.9822	0.1496	**ESE**	-10.5800	0.3020
miRNA	7.3640	0.3960	**ESS**	-6.5410	0.4637
Transcript	4.0052	0.0567*	**Splicing**	-3.0343	0.8299
RIT	-1.9769	0.8532	**Coding**	20.9997	0.6337

Table 6. Coefficient for a linear model with significant noncoding motif proportions and coding mutations as explanatory variables. *P < 0.1; **P < 0.05

having a p-value of 0.09. Nonetheless, using this list, we can rank the motifs that may significantly affect individual or multiple cancer pathways. From these analyses, I noticed that the intron splicing enhancers are not only associated with the cell cycle pathway but also appear to affect other pathways, including the dorsoventral axial pathway and the progesterone mediated oocyte maturation pathway. Similarly, motifs such as the exon splicing enhancers and exon splicing silencers appear to affect the score of at least one of the pathways. Thus, in almost all of these pathways, I detected a splicing-related motif that has a significant effect on the pathway score. This suggests that the splicing-related motifs may have a key role in the relation between noncoding mutations and cancer.

Pathway	FDR-adjusted Equation p-value	Significant Motifs (p < 0.05)
Cell Cycle	0.09423408	Transcript, ISE
Dorsoventral Axial Formation	0.09423408	ncRNA, longStems, RIT, funcRNA, ESE, ISE, Coding
Pathogenic E. Coli Infection	0.10729589	RIT, ESS
Progesterone Mediated Oocyte Maturation	0.14094723	ncRNA, Transcript, ESE, ESS, ISE
Taste Transduction	0.16675849	Splicing, longStems, miRNA, ESS

Table 7. Table of pathways with the lowest equation p-values after FDR correction and their significant motifs.

CONCLUSIONS

1. Discussion

I developed computational methods to identifying noncoding mutations that may impact cancer development. In this study, I analyzed whole genome sequencing data for 28 breast cancer patients. Based on the analysis, I identified a number of functional RNA motifs that carried a significantly different number of mutations compared to what would be expected at random. As noted previously, much of cancer mutation research has focused on the relation between coding mutations and cancer; thus, by focusing on identifying functional RNA motifs that are significantly differentially affected by noncoding mutations, I have taken steps to clarify the connection between these mutations and cancer. Furthermore, I conducted additional analyses correlating the noncoding mutations with cancer-related pathways. From this, I observed that splicing-related motifs may have a large impact on function. This indicates that previous methods that have focused primarily on transcriptional elements have missed a large portion of the genome that is enriched for possible drivers. These findings suggest that the current analysis may be useful for identifying regions of the genome that are significant to cancer development based on function RNA motifs.

A number of interesting observations were made from this study. First, noncoding mutations in splicing-related functional RNA motifs, including intron splicing enhancers, exon splicing silencers, and exon splicing enhancers were significantly linked to breast cancer. More importantly, these types of motifs were consistently found to be significantly associated with cancer-related pathways, such

as the cell cycle pathway, indicating that they are functional. This finding is strongly supported since splicing errors can lead to early termination, alternative splicing, and instability of the mRNA, rendering loss of expression or changes in the function of the proteins that play critical roles in the cancer-related pathways. Recently, it was reported that, based on RNA sequencing data, alternative exon usage was widely present in breast cancer samples and that some of the alternatively spliced products are specific to breast cancer subtypes (Bjørklund et al., 2017). Therefore, alternative splicing can be crucial contributors to oncogenesis and specific splicing events on specific genes may be associated with particular breast cancer subtypes. This is consistent with the finding presented here that mutations on splicing-related RNA motifs are associated with breast cancer. Similarly, transcriptional regulatory motifs are also important for producing mRNAs, which in turn produce protein products. Mutations in these motifs will likely alter protein expression and thereby impact cancer development. Indeed, I found that noncoding mutations in the transcriptional regulatory motifs tended to be associated with the cell cycle pathway, though the p-value is borderline significant.

Second, I found that in regions predicted to affect RNA processing that the number of mutations detected in the tumor samples was less than would be expected at random. This suggests that these motifs are less tolerant toward mutations relative to other regions. One potential explanation is that these motifs are critical to cell survival such that genetic variations are detrimental to the cell. This could be specific to cancer cells or possibly healthy cells in general. To distinguish between these possibilities, we would need to examine genetic variations

in these motifs in normal cell samples. If these motifs also carry fewer variants in normal samples, then they are likely to be crucial toward general cell survival. On the other hand, if the motifs are negatively enriched by noncoding mutations only in the cancer samples, then those motifs may be specifically important for supporting cancer growth.

Third, I observed that certain cancer-related pathways appeared to be significantly associated with particular identified significant motifs in breast cancer, such as the KEGG cancer pathways and the ISE and transcriptional regulatory motifs. The reason for this is not clear. One possibility is that the specific functions defined by the motifs are particularly crucial for RNA processing, transcriptional regulation, or other functions of at least one gene in the pathway. In this case, there may be specific motifs that bear recurrent mutations, while the affected genes play critical roles in the pathway's function. Detailed examination of the motifs with recurrent mutations in all samples would help to clarify the situation. Alternately, it is also possible that certain pathways just contain more genes compared to other pathways, which would then affect cell growth and proliferation when mutated.

Finally, due to the small sample size of the study, its statistical power is limited, which means that some of the observed associations may be random. Increasing the sample size would likely improve the quality of the analysis and help verify validity of the associations.

2. Future Directions

There are a number of improvements that can be made to this study. Among them, the most important task would be to obtain more samples from the TCGA

database, which would increase the power of our statistics. Currently, we have analyzed 28 samples of the 118 whole genome samples available for breast cancer from the TCGA. Additionally, other projects have generated more data featuring WGS analysis, such as the ICGC PanCancer Analysis of Whole Genomes (PCAWG) study. Having more breast cancer samples would also allow us to categorize them by different subtypes, which could uncover more specific mechanisms of noncoding mutations. These mechanisms may also be clarified by analyzing the motifs by name (e.g., eukaryotic-type signal recognition particle), rather than type (e.g. functional RNA sequences), which can better identify the specific RNA processes that are significantly associated with noncoding mutations.

Various improvements can be made to the filters and algorithms used throughout the study, which would allow us to improve and verify the data and thus our conclusions. For example, using other callers aside from MuTect would allow us to cross-check the results, thus improving the accuracy of detected mutations. Adjusting the filters that define the set of noncoding mutations would allow us to identify relevant mutations more reliably. While the method for generating the background data in Aim 1 accounts for the location-based filters of the noncoding mutations, it does not consider the other filters such as the minimum read depth, which may introduce a bias because of how the read depth can be affected by things such as GC content. Failing to take this filter into account may cause the random selection to select regions of the genome that are underrepresented among the actual variants. It may also be appropriate to explore alternate models for the regression analysis in Aim 2. Developing a different linear or nonlinear model may

improve the analysis and give more insight into which pathways are strongly impacted by the presence of noncoding mutations on RNA motifs. Finally, we could analyze pathways from sources other than KEGG to get more information on the functional impact of noncoding mutations.

Based on the results of the study, splicing-related motifs appear to be the most significant when affected by noncoding mutations in the context of cancer. These results can be verified by analyzing the samples that were not used in this study and checking if the findings are consistent. The presence of splicing-related motifs can also be checked by analyzing RNA-seq data to verify if alternative splicing can be found in predicted areas. If splicing-related motifs remain relevant after verification, then the next step would be to further investigate the recurrence of alternative splicing in tumor samples and their importance to cancer development.

APPENDIX

1. Scripts

1.1 runbetsy.sh

Runs Betsy on chosen folder (`caseNum`). Multiple BAM files from different samples can be processed at once. Tumor samples should be named `tumor.case##.bam` with matched normals named `normal.case##.bam`. Processing tasks include realigning via BWA-MEM, marking duplicates, checking for read groups, realigning indels, sorting by coordinates, and indexing folders.

1.2 runMutect.sh

Runs MuTect on selected files. Tumor sample should be named `tumor.case##.bam` with matched normal named `normal.case##.bam`. To analyze a sample more quickly, the script is run per chromosome in parallel.

1.3 procGenes.sh

Processes MuTect output and annotates identified mutations via ANNOVAR. Outputted files will also include information from the MuTect output, such as read depth and MuTect's judgment on the variant.

1.4 variantFilter.py

Filters variants based on criteria as listed in the Methods section. Additionally separates the variants into coding and noncoding.

1.5 variantGet.py

Generates a hash for variants from ANNOVAR output files. Hashes are generated from the chromosome, reference allele, alternate allele, and location of the variant.

1.6 run_regrna.py

Processes DNA sequences through the RegRNA 2.0 web program. The *mechanize* library for Python is used to access the web. Results for each sequence are stored by chromosome. The script then identifies the RNA motifs affected by each variant from a list and outputs them to a file. The table consisting of the number of variants affecting a particular motif for each sample can be created.

1.7 hgTables_slim.py

Generates a random set of locations for the background data in Aim 1. Genome and gene information is taken from the UCSC Genome Browser. Valid segments are selected based on criteria used for selecting variants (e.g., non-exonic, location < 25,000 bps from the end of a gene). Random points are then uniformly randomly selected from all valid positions.

1.8 regression2.R

Conducts linear regression on coding and noncoding mutations relative to the pathway score, as described in Methods for Aim 2.

1.1 runbetsy.sh

```
#!/bin/bash

caseNum=case28
refFile=hsa19/Homo_sapiens_assembly19.fasta
A=Mills_and_1000G_gold_standard

python ~/changlab/Betsy/scripts/betsy_run.py
 --network_png ~/${caseNum}_network.png
 --input BamFolder --input_file ~/data/$caseNum/bamfiles/
 --dattr BamFolder.aligner=bwa_mem
 --input ReferenceGenome --input_file ~/data/$refFile
 --output BamFolder
 --output_file ~/data/$caseNum/bamfile_proc/
 --dattr BamFolder.aligner=bwa_mem
 --dattr BamFolder.duplicates_marked=yes
 --dattr BamFolder.has_read_groups=yes
 --dattr BamFolder.indel_realigned=yes
 --dattr BamFolder.sorted=coordinate
 --dattr BamFolder.indexed=yes
 --dattr BamFolder.base_quality_recalibrated=yes
 --mattr realign_known_sites1=~/v/$A.indels.hg19.sites.vcf.gz
 --mattr
   realign_known_sites2=~/v/1000G_phase1.indels.hg19.sites.vcf
   .gz
 --mattr recal_known_sites1=~/v/$A.indels.hg19.sites.vcf.gz
 --mattr
   recal_known_sites2=~/v/1000G_phase1.indels.hg19.sites.vcf.g
   z
 --mattr recal_known_sites3=~/v/dbsnp_138.hg19.vcf.gz
 --num_cores 40 –run
```

1.2 runMutect.sh

```
#!/bin/bash

caseNum=28
thisCase=case$caseNum

mkdir ~/data/$thisCase/processed
samtools idxstats
~/data/$thisCase/bamfile_proc/tumor.$thisCase.bam |
 cut -f 1 > ~/data/$thisCase/processed/listChr.txt

i=1
```

```
thisChr=$(sed "$i"'q;d'
~/data/$thisCase/processed/listChr.txt)
flag=true
fileString=~/data/$thisCase/processed/chr$thisChr.out

while [ "$flag" = true ] && [ $i -lt $(wc -l
 ~/data/$thisCase/processed/listChr.txt | cut -f 1 -d ' ') ]
do
   if [ -f $fileString ]
   then
   ((i++))
           thisChr=$(sed "$i"'q;d'
                  ~/data/$thisCase/processed/listChr.txt)
   fileString=~/data/$thisCase/processed/chr$thisChr.out
   else
     flag=false
   fi
done

if [ $i -lt $(wc -l ~/data/$thisCase/processed/listChr.txt |
cut -f 1
 -d ' ') ]
then
   java -Xmx2g -jar ~/data/mutect-src/mutect/target/mutect-
1.1.7.jar
       --analysis_type MuTect
       --reference_sequence
         ~/data/hsa19/Homo_sapiens_assembly19.fasta
       --cosmic ~/data/b37_cosmic_v54_120711.vcf
       --dbsnp ~/data/dbsnp_132_b37.leftAligned.vcf
       --intervals $thisChr
       --input_file:normal
         ~/data/$thisCase/bamfile_proc/normal.$thisCase.bam
       --input_file:tumor
         ~/data/$thisCase/bamfile_proc/tumor.$thisCase.bam
       --out $fileString
fi
```

1.3 procGenes.sh

```
#!/bin/bash

for i in 28
do
   caseNum=case$i
   caseFold=~/data/$caseNum/processed/info
```

```
   cat ~/data/$caseNum/processed/chr* >
      ~/data/mutectFull/out.$caseNum.out
   rm ~/data/$caseNum/processed/chr*

   cut -f 1,2 ~/data/mutectFull/out.$caseNum.out >
      ~/data/$caseNum/processed/part1
   cut -f 2 ~/data/mutectFull/out.$caseNum.out >
      ~/data/$caseNum/processed/part2
   cut -f 4,5,16,26,27,38,39,51
      ~/data/mutectFull/out.$caseNum.out >
      ~/data/$caseNum/processed/part3
   mkdir ~/data/$caseNum/processed/info
   paste ~/data/$caseNum/processed/part1
        ~/data/$caseNum/processed/part2
        ~/data/$caseNum/processed/part3 >
        ~/data/$caseNum/processed/info/$caseNum.anvinput
   rm ~/data/$caseNum/processed/part*
   ~/data/mutect-src/annovar/annotate_variation.pl
      -out ~/data/$caseNum/processed/info/$caseNum.anvoutput
      -build hg19
      ~/data/$caseNum/processed/info/$caseNum.anvinput
      ~/data/mutect-src/annovar/humandb

   echo $caseNum complete

Done
```

1.4 variantFilter.py

```
import sys
import os

for i in range(28):
   thisCase = 'case' + str(i+1)
   fCase = open('data/' + thisCase + '/processed/info/' +
           thisCase + '.anvoutput.variant_function')

   codMuts = []
   regMuts = []

   for line in fCase:
       lineList = line.split('\t')

       if lineList[12].strip() == "KEEP" and
          int(lineList[7].strip())>30:

           mutType = [x.strip() for x in
```

```
                    lineList[0].split(';')]
            geneNom = lineList[1].strip()
            if "exonic" in mutType:
                codMuts.append(line)
            elif "intergenic" in mutType:
                if ',' in geneNom:
                    interGenes = geneNom.split(',')
                    distList = []
                    for aGene in interGenes:
                        temp =
                          aGene[aGene.find('=')+1:aGene.find(')
                                ')]
                        if "NONE" not in temp:
                            distList.append(int(temp))
                    if any([x < 25000 for x in distList]):
                        regMuts.append(line)
            else:
                regMuts.append(line)

    fCodOut = open('data/' + thisCase + '/processed/info/' +
              thisCase + ".codMuts.filtered.txt", 'w')
    for line in codMuts:
        fCodOut.write(line)

    fRegOut = open('data/' + thisCase + '/processed/info/' +
              thisCase + ".regMuts.filtered.txt", 'w')
    for line in regMuts:
        fRegOut.write(line)
```

1.5 variantGet.py

```
import sys
import os

for i in range(28):
    thisCase = 'case' + str(i+1)
    fCase = open('data/' + thisCase + '/processed/info/' +
            thisCase + '.regMuts.filtered.txt')

    varList = []
    for line in fCase:
        lineList = line.split('\t')
        mutChr = lineList[2].strip()
        mutLoc = lineList[3].strip()
        mutRef = lineList[5].strip()
        mutAlt = lineList[6].strip()
```

```
        varList.append(mutChr + mutRef + mutAlt + mutLoc)

    fOut = open('data/' + thisCase + '/processed/info/' +
           thisCase + '.regVars.txt', 'w')
    for aVar in varList:
        fOut.write(aVar + '\n')
```

1.6 run_regrna.py

```
import sys, os
import re
from urllib2 import HTTPError
import mechanize
from genomicode import genomelib
import random, time, shutil

assert mechanize.__version__ >= (0,0,6,"a")

def readRegions(thisType='real'):

    if thisType == 'fake':
        fOpen=open('fakeRegionsCovered.txt')
    else:
        fOpen = open('regionsCovered.txt')
    header = fOpen.readline()

    chrDict = {}
    for line in fOpen:
        lineList = line.split('\t')
        thisChr = lineList[0].strip()
        thisPos = int(lineList[1].strip())
        thisLen = int(lineList[2].strip())
        if thisChr not in chrDict:
            chrDict[thisChr] = []
        chrDict[thisChr].append((thisPos, thisLen))
    fOpen.close()
    return chrDict

def updateChrDict():
    global gloChrDict
    gloChrDict = readRegions()

def getPos(pos):
    if list(pos)==pos:
        return pos
    else:
        thisLen = 0
```

```
        if '.' in pos:
            tempPos = pos.split('.')
            thisChr = tempPos[0]
            thisPos = tempPos[1]
            if len(tempPos)>2:
                thisLen = tempPos[2]
        else:
            thisChr = pos[0:2]
            thisPos = pos[4:]
            if any(x in thisChr for x in ['A', 'T', 'C', 'G',
               'N']):
                thisChr = pos[0:1]
                thisPos = pos[3:]

        return [thisChr, thisPos, thisLen]

def fillForm(mech, fasta):
   mech.select_form(nr=0)
   mech['S1']=fasta
   mech['tfbs'] = ['ON']
   mech['SplicingSite']=['ON']
   mech['SplicingMotif']=['ON']
   mech['Polya']=['ON']
   mech['RBSfinder']=['ON']
   mech['rho']=['ON']
   mech['UTRsite']=['ON']
   mech['AUrich']=['ON']
   mech['RNAediting']=['ON']
   mech['RiboSW']=['ON']
   mech['ERPIN']=['ON']
   mech['Rfam']=['ON']
   mech['LongStem']=['ON']
   mech['fRNAdb']=['ON']
   mech['miRNA']=['ON']
   mech['ncRNA']=['ON']
   mech['GCratio']=['ON']
   mech['accessibility']=['ON']

   return mech

def fillFormFix(pos, ball):
   mech = setupBrowser()
   mech.select_form(nr=0)
   thisFa = generateFa(pos, ball)
   mech['S1']=thisFa
   mech['SplicingSite']=['ON']
   mech['SplicingMotif']=['ON']
```

```
    mech.submit()
    print "Form fix submitted"

    tempList = thisFa.split('\n')
    fileNom=tempList[0][1:]
    urls = [link.absolute_url for link in
           mech.links(url_regex=re.compile('\.all\.result'))]
    url = urls[0]
    filename='/home/kzhu/data/RegRNA/regrna_resultstemp/'+fileN
              om+'.txt'
    f=open(filename, 'wb')
    print filename
    r = mech.open(url)
    while 1:
        data = r.read(1024)
        if not data: break
        f.write(data)
    f.close()

    if checkIfFileExists(fileNom, 'min'):
        f=open(filename, 'r')
        fOut = open('/home/kzhu/data/RegRNA/regrna_results/' +
               fileNom + '.txt', 'a')
        for line in f:
            if any([x in line for x in ['(ESE)', '(ESS)',
               '(ISE)', '(ISS)']]):
                fOut.write(line)
        fOut.close()
        f.close()
        os.remove(filename)

def tempRun():
  #fixes an error from a previous RegRNA run
    fixedList = []
    fOpen = open('fixedList.txt')
    for line in fOpen:
        fixedList.append(line.strip())
    fOpen.close()
    fOpen = open('fixedList.txt', 'a')
    fiList = os.listdir('regrna_results')
    for i in fiList:
        if i not in fixedList:
            fillFormFix(i.strip('.txt'), 10000)
            fOpen.write(i + '\n')
            fixedList.append(i)
            print 'Fix complete. Waiting 30 seconds...'
            time.sleep(30)
```

```
    fOpen.close()

def generateFa(pos='', ball=10000):

    thisPos = getPos(pos)

    thisChr = thisPos[0]
    thisStart = thisPos[1]
    thisLen = int(thisPos[2]) if (int(thisPos[2])>0) else ball

    thisSeq = genomelib.get_sequence(thisChr, int(thisStart),
              int(thisLen))
    thisOut = '>' + thisChr + '.' + thisStart + '.' +
               str(thisLen)
    counter = 0
    for i in thisSeq:
        if counter % 50 == 0:
            thisOut += '\n'
        thisOut +=i
        counter +=1

    return thisOut

def setupBrowser():
    regrna = mechanize.Browser()
    regrna.set_handle_robots(False)
    try:
        regrna.open("http://regrna2.mbc.nctu.edu.tw/detection.h
        tml")
    except HTTPError, e:
        return ''
    return regrna

def fillAndSub(pos='', ball=10000):
    thisFa = generateFa(pos, ball)
    temp = setupBrowser()
    if temp:
        mech = fillForm(temp, thisFa)
        mech.submit()

        print "Form Submitted"

        tempList = thisFa.split('\n')
        fileNom = tempList[0][1:]

        urls=[link.absolute_url for link in
              mech.links(url_regex=re.compile(r"\.all\.result"))
```

```
            ]
        url = urls[0]
        filename =
           '/home/kzhu/data/RegRNA/regrna_resultstemp/'+fileNom
           +'.txt'
        f=open(filename, 'wb')
        print filename
        r=mech.open(url)
        while 1:
            data=r.read(1024)
            if not data: break
            f.write(data)
        f.close()
    else:
        f = open('redo.txt', 'a')
        f.write(pos + '\n')
        f.close()

def checkIfCovered(pos):
    thisPos = getPos(pos)
    thisChr = thisPos[0]
    thisStart = str(thisPos[1])

    thesePos = gloChrDict[thisChr]
    isCov = any( [int(thisStart) >= (x[0]+200) and
      int(thisStart) <= (x[0]+x[1]-200) for x in thesePos] )
    return isCov

def getCover(pos):
    thisPos = getPos(pos)
    thisChr = thisPos[0]
    thisStart = thisPos[1]

    thesePos = gloChrDict[thisChr]
    findCov = [int(thisStart) >= (x[0]+200) and int(thisStart)
               <= (x[0]+x[1]-200) for x in thesePos]
    thisInd = findCov.index(True)

    thisVal = thesePos[thisInd]
    thisHash = thisChr + '.' + str(thisVal[0]) + '.' +
str(thisVal[1])

    return thisHash

def checkIfFileExists(aHash, level="full"):
    theseFiles = os.listdir('regrna_results')
    if level=='full':
```

```
        theseFiles.extend(os.listdir('regrna_resultstemp'))
    return ((aHash+'.txt') in theseFiles)

def runTest():
    fOpen = open('fakePosList4.reg.txt')
    posList = []
    for line in fOpen:
        posList.append(line.strip())
    fOpen.close()

    blackList = ['7.158122530.10000', '14.19281652.10000',
      '14.19293547.10000', '19.6769083.10000',
      '14.19262029.10000', '14.19307662.10000',
      '14.19292736.10000', '14.19272995.10000',
      '14.19307640.10000', '14.19292291.10000']
#certain segments will cause RegRNA to stall (not sure why).
 for simplicity, just ignore them

    for i in posList:
        if i and checkIfCovered(i):
            thisHash = getCover(i)
            if not checkIfFileExists(thisHash) and not thisHash
               in blackList:

                print i, thisHash, checkIfFileExists(thisHash)
                fillAndSub(thisHash)
                time.sleep(30)

def updateMotif():
    fPosList = open('checkedResults.txt', 'r')
      ###checklist to avoid reexamining sequences
    checkList = []
    for line in fPosList:
        checkList.append(line.strip())
    fPosList.close()

    fPosList = open('checkedResults.txt', 'a')
    theseFiles = os.listdir('regrna_results')
      ###Obtain list of files containing results for analyzed
      ###sequences
    chrMotList = os.listdir('motifInfo')

############################### filtering lists for
categorizing
    fRan = open('adjRandomPosListAll.txt')
    header = fRan.readline()
    randList = []
```

```
    for line in fRan:
        randList.append(line.strip().replace('\t', '.'))
    fRan.close()

    fFake = open('fakeRegionsCovered.txt')
    header = fFake.readline()
    fakeList = []
    for line in fFake:
        fakeList.append(line.strip().replace('\t', '.'))
    fFake.close()

    fFake2 = open('fakeRegionsCovered2.txt')
    header = fFake2.readline()
    fake2List = []
    for line in fFake2:
        fake2List.append(line.strip().replace('\t', '.'))
    fFake2.close()
################################### end filtering lists

    for i in theseFiles:
        if i not in checkList:

            aList = i.split('.')
            thisChr = aList[0]
            thisPos = aList[1]
            thisLen = aList[2]
            thisSeq = 'variant'

            if i in [x + '.txt' for x in randList]:
                thisSeq = 'random'
            elif i in [x + '.txt' for x in fakeList]:
                thisSeq = 'gene-based'
            elif i in [x + '.txt' for x in fake2List]:
                thisSeq = 'gene-based2'

            outFile = 'chr' + thisChr + '.txt'

            if outFile not in chrMotList:
                fOut = open('motifInfo/' + outFile, 'w')
                fOut.write('Chromosome\tStart
                            Position\tSequence
                            Type\tLength\tMotif Type\tMotif
                            Name')
                fOut.close()
                chrMotList.append(outFile)

            fOut = open('motifInfo/' + outFile, 'a')
```

```
            fOpen = open('regrna_results/' + i)
            header = fOpen.readline()
            print i

            for line in fOpen:
                if line.strip():
                    lineList = line.split('\t')
                    motType = lineList[0].strip()
                    motNom = lineList[1].strip()
                    motPos = lineList[2].strip()

                    posSplit = motPos.split('~')
                    motStart = int(posSplit[0].strip())-1
                    motLen = int(posSplit[1].strip())-motStart

                    thisStr = '\n' + thisChr + '\t' +
                              str(int(thisPos) + motStart) +
                              '\t' + thisSeq + '\t' +
                              str(motLen) + '\t' + motType +
                              '\t' + motNom

                    fOut.write(thisStr)

            fOpen.close()
            fOut.close()

            checkList.append(i)
            fPosList.write(i + '\n')

    fPosList.close()

def motPurge():
    fiList = os.listdir('motifInfo')
    for i in fiList:
        os.remove('motifInfo/' + i)
    fRes = open('checkedResults.txt', 'w')
    fRes.close()

def addToPosList(pos):
    global gloChrDict
    thisPos = getPos(pos)
    thisChr = str(thisPos[0])
    thisStart = str(thisPos[1])
    thisLen = int(thisPos[2]) if (int(thisPos[2]) > 0) else
              10000
```

```
    fOpen = open('regionsCovered.txt', 'a')

    if thisChr not in gloChrDict:
        gloChrDict[thisChr] = []
    gloChrDict[thisChr].append((int(thisStart), thisLen))

    fOpen.write('\n' + thisChr + '\t' + thisStart + '\t' +
                str(thisLen))
    fOpen.close()

def addTrueRandom(aNum):
    i = 0
    while i < aNum:
        thisPos = getPos('')
        while checkIfCovered(thisPos):
            thisPos = getPos('')
        i+=1
        addToPosList(thisPos)

def addAdjRandom(aNum):
    randChr = [random.choice(gloChrDict.keys()) for i in
               range(aNum)]
    fOut = open('adjRandomPosList5.txt', 'w')
    fOut.write('Chromosome\tStart Position\tLength')
    for i in randChr:
        posList = [x[0] for x in gloChrDict[i]]
        randPos = random.choice(posList)
        thisPos = []
        while not thisPos:
            posA = [(i+'.'+str(randPos-11000+x)+'.10000') for x
                    in [-200, 0, 200]]
            posB = [(i+'.'+str(randPos+11000+x)+'.10000') for x
                   in [-200, 0, 200]]
            tempPos = [any([checkIfCovered(x) for x in posA]),
                       any([checkIfCovered(x) for x in posB])]
            if not any(tempPos):
                thisPos = random.choice([posA[1], posB[1]])
            elif not all(tempPos):
                thisPos = posB[1] if tempPos[0] else posA[1]
            else:
                randPos = random.choice(posList)
                thisPos = []
        addToPosList(thisPos)
        thisPos = getPos(thisPos)
        fOut.write('\n' + i + '\t' + str(thisPos[1]) + '\t' +
                   str(thisPos[2]))
    fOut.close()
```

```
def runRandTest():
   fOpen = open('adjRandomPosListAll.txt')
   header = fOpen.readline()
   posList = []
   for line in fOpen:
       lineList = line.split('\t')
       posList.append(lineList[0].strip() + '.' +
         lineList[1].strip() + '.' + lineList[2].strip())
   fOpen.close()
   for i in posList:
       if not checkIfFileExists(i):
           fillAndSub(i)

def posScan(pos, mType='mut', motDict=''):
   thisPos = getPos(pos)
   thisChr = thisPos[0]
   thisStart = int(thisPos[1])

   chrFile = 'chr' + thisChr + '.txt'
   existFiles = os.listdir(mType + 'Motifs/')

   if checkIfCovered(thisPos) and chrFile in
      os.listdir('motifInfo') and not (pos+'.txt' in
      existFiles):

       thisHash = getCover(thisPos)
       if checkIfFileExists(thisHash, 'min'):

           fOut = open(mType + 'Motifs/' + pos + '.txt', 'w')
           fOut.write('Name\tChromosome\tStart Position\tMotif
                      Type\tMotif Names')
           strPrefix = '\n' + pos + '\t' + thisChr + '\t' +
                      str(thisStart)

           if not motDict:
               varMots = procChrMotifs(chrFile)
           else:
               varMots = motDict[thisChr]

           for aType in varMots.keys():
               for aStart in varMots[aType].keys():
                   isCov = [thisStart >= int(aStart) and
                            thisStart <= (int(aStart) + x[0])
                            for x in varMots[aType][aStart]]
                   if any(isCov):
                       indList = [x for x,val in
```

```
                                    enumerate(isCov) if val]
                        for i in indList:
                            thisStr = strPrefix + '\t' + aType
                                      + '\t'
                            for motNom in [x[1] for x in
                               varMots[aType][aStart]]:

                                thisStr += motNom + ';'

                            thisStr = thisStr.strip(';')
                            fOut.write(thisStr)
            fOut.close()

def posScanAll():
   motDict = gloMotDict

   print(motDict.keys())

   mutPosList = []
   fPos = open('poslist3.txt')
   for line in fPos:
       if line.strip():
           mutPosList.append(line.strip())
   fPos.close()

   fakePosList = []
   fFake = open('fakePosList.reg.buffer.txt')
   for line in fFake:
       fakePosList.append(line.strip())
   fFake.close()

   fake2PosList = []
   fFake2 = open('fakeMutList4.txt')
   for line in fFake2:
       fake2PosList.append(line.strip())
   fFake2.close()

   fake3PosList = []
   fFake3 = open('fakeMutList.fakeonly.reg.txt')
   for line in fFake3:
       fake3PosList.append(line.strip())
   fFake3.close()

   fakeIIPosList = []
   fFakeII = open('fakePosList2.reg.txt')
   for line in fFakeII:
       fakeIIPosList.append(line.strip())
```

```
fFakeII.close()

fake4PosList = []
fFake4 = open('fakePosList4.reg.txt')
for line in fFake4:
    fake4PosList.append(line.strip())
fFake4.close()

if 0:
    for i in mutPosList:
        temp = getPos(i)
        thisChr = temp[0]
        if thisChr not in motDict:
            motDict[thisChr] = procChrMotifs('chr' +
                               thisChr + '.txt')
        posScan(i, 'mut', motDict)
    print('Variants completed')
else:
    print('Variants skipped')

if 0:
    for i in fakePosList:
        temp = getPos(i)
        thisChr = temp[0]
        if thisChr not in motDict:
            motDict[thisChr] = procChrMotifs('chr' +
                               thisChr + '.txt')
        posScan(i, 'fake', motDict)
    print('Random A completed')
else:
    print('Random A skipped')

if 0:
    for i in fake2PosList:
        temp = getPos(i)
        thisChr = temp[0]
        if thisChr not in motDict:
            motDict[thisChr] = procChrMotifs('chr' +
                               thisChr + '.txt')
        posScan(i, 'fake2', motDict)
    print('Random B completed')
else:
    print('Random B skipped')

if 0:
    for i in fake3PosList:
        temp = getPos(i)
```

```
            thisChr = temp[0]
            if thisChr not in motDict:
                motDict[thisChr] = procChrMotifs('chr' +
                                   thisChr + '.txt')
            posScan(i, 'fake3', motDict)
        print('Random C completed')
    else:
        print('Random C skipped')

    if 0:
        for i in fakeIIPosList:
            temp = getPos(i)
            thisChr = temp[0]
            if thisChr not in motDict:
                motDict[thisChr] = procChrMotifs('chr' +
                                   thisChr + '.txt')
            posScan(i, 'fakeII', motDict)
        print('Random D completed')
    else:
        print('Random D skipped')

    if 1:
        for i in fake4PosList:
            temp = getPos(i)
            thisChr = temp[0]
            if thisChr not in motDict:
                motDict[thisChr] = procChrMotifs('chr' +
                  thisChr + '.txt')
            posScan(i, 'fake4', motDict)
        print('Random E completed')
    else:
        print('Random E skipped')

def procChrMotifs(aFile, seqMat=False):
    fOpen=open('motifInfo/' + aFile)
    header = fOpen.readline()

    motDict = {}

    for line in fOpen:
        if line.strip():
            lineList = line.split('\t')
            thisChr = lineList[0].strip()
            thisStart = int(lineList[1].strip())
            thisLen = int(lineList[3].strip())
            thisType = lineList[4].strip()
            thisNom = lineList[5].strip()
```

```
            thisSeq = lineList[2].strip()

            if seqMat:
                if thisSeq not in motDict:
                    motDict[thisSeq] = {}
                if thisType not in motDict[thisSeq]:
                    motDict[thisSeq][thisType] = {}
                if thisStart not in motDict[thisSeq][thisType]:
                    motDict[thisSeq][thisType][thisStart] = []
                motDict[thisSeq][thisType][thisStart].append((t
                  hisLen, thisNom))
            else:
                if thisType not in motDict:
                    motDict[thisType] = {}
                if thisStart not in motDict[thisType]:
                    motDict[thisType][thisStart] = []
                motDict[thisType][thisStart].append((thisLen,
                  thisNom))
    fOpen.close()

    return motDict

def arrangeSegs(segList):
    segList.sort(key=lambda x: x[0])
    tempList = []
    tempTup = (0,0)

    for i in segList:
        if i[0] > sum(tempTup):
            if not tempTup[1] == 0:
                tempList.append(tempTup)
            tempTup = i
        elif sum(i) > sum(tempTup):
            thisLen = (i[0] - tempTup[0]) + i[1]
            tempTup = (tempTup[0], thisLen)
    return tempList

def motBaseInfo(aMot=''):
    fiList = os.listdir('motifInfo')
    chrList = filter(lambda x: 'chr' in x, fiList)
    motDict = {}

    for aChr in chrList:
        thisDict = procChrMotifs(aChr, True)
        thisChr = aChr.strip('.txt').strip('chr')
        for seqType in thisDict.keys():
```

```
            if seqType not in motDict:
                motDict[seqType] = {}
            chrDict = thisDict[seqType]
            if not aMot:
                for i in chrDict.keys():
                    if i not in motDict[seqType]:
                        motDict[seqType][i] = {}
                    tempMots = []
                    tempTup = (0,0)

                    thesePos = chrDict[i].keys()
                    thesePos.sort(key=int)

                    for j in thesePos:
                        tempLen = max([x[0] for x in
                                  chrDict[i][j]])
                        if j > sum(tempTup):
                            if not tempTup[1] == 0:
                                tempMots.append(tempTup)
                            tempTup = (j, tempLen)
                        else:
                            thisLen = (j - tempTup[0]) +
                                      tempLen
#start position of new section minus start position of
existing plus length of new section

                            tempTup = (tempTup[0], thisLen)

                    motDict[seqType][i][thisChr] = tempMots

            elif aMot in chrDict.keys():
                aMotDict = chrDict[aMot]
                tempList = []
                dummy = [tempList.extend(y) for y in
                        [aMotDict[x] for x in aMotDict.keys()]]
                motNomList = [x[1] for x in tempList]
                motNomList = list(set(motNomList))
                for i in motNomList:
                    thesePos = filter(lambda x: any([y[1]==i
                               for y in aMotDict[x]]),
                               aMotDict.keys())
                    thesePos.sort(key=int)

                    if i not in motDict[seqType]:
                        motDict[seqType][i]={}
                    tempMots = []
                    tempTup = (0,0)
```

```
                for j in thesePos:
                    tempLen = max([x[0] for x in
                              filter(lambda y: y[1]==i,
                              aMotDict[j])])
                    if j > sum(tempTup):
                        if not tempTup[1] == 0:
                            tempMots.append(tempTup)
                        tempTup = (j, tempLen)
                    else:
                        thisLen = (j - tempTup[0]) +
                                   tempLen
                        tempTup = (tempTup[0], thisLen)
                motDict[seqType][i][thisChr] = tempMots

tempList = [str(x) for x in range(1,23)]
tempList.append('X')
lenDict = {}
strDict = {}

for seqType in motDict.keys():
    lenDict[seqType] = {}
    tempDict = {}
    aDict = motDict[seqType]

    tempMots = filter(lambda x: not len(aDict[x].keys()) ==
               len(tempList), aDict.keys())
    if tempMots:
        for i in tempMots:
            theseChr = filter(lambda x: x not in
                       aDict[i].keys(), tempList)
            for j in theseChr:
                motDict[seqType][i][j] = []

    for i in tempList:
        tempSegs = []
        dummy = [tempSegs.extend(aDict[y][i]) for y in
                 aDict.keys()]
        tempDict[i] = arrangeSegs(tempSegs)
    motDict[seqType]['any'] = tempDict

    motifStr = ''
    for i in aDict.keys():
        motifStr += '\t' + i
        lenDict[seqType][i] = {}
        lenDict[seqType][i]['tot'] = 0
        for j in aDict[i].keys():
```

```
                tempSum = sum([x[1] for x in aDict[i][j]])
                lenDict[seqType][i][j] = tempSum
                lenDict[seqType][i]['tot'] += tempSum
        strDict[seqType] = motifStr

    tempList.append('tot')

    tempStr = ''
    if aMot:
        if '(' in aMot:
            tempStr = aMot[(aMot.index('(') +
                      1):aMot.index(')')]
        else:
            tempStr = aMot[0:aMot.index(' ')]

    for seqType in motDict.keys():
        fOut = open('motifBaseInfo.'+seqType+tempStr+'.txt',
               'w')
        fOut.write(strDict[seqType] + '\n')
        for j in tempList:
            fOut.write(j)
            tempDict = lenDict[seqType]
            for i in tempDict.keys():
                thisStr = str(tempDict[i][j]) if j in
                          tempDict[i].keys() else '0'
                fOut.write('\t' + thisStr)
            fOut.write('\n')

        fOut.close()
    return lenDict

def mutBaseInfo(prefix='mut', aMot=''):

    fiList = os.listdir(prefix + 'Motifs')     ###

    motList = []
    posDict = {}

    for aFile in fiList:
        thisPos = aFile.strip('.txt')
        posDict[thisPos] = []
        fOpen = open(prefix + 'Motifs/' + aFile)     ###
        header = fOpen.readline()
        for line in fOpen:
            lineList = line.split('\t')
            motType = lineList[3]
            motNom = lineList[4].strip().split(';')
```

```
            if aMot and motType==aMot:
                for aNom in motNom:
                    if aNom not in motList:
                        motList.append(aNom)
                    if aNom not in posDict[thisPos]:
                        posDict[thisPos].append(aNom)
            elif not aMot:
                if motType not in motList:
                    motList.append(motType)
                if motType not in posDict[thisPos]:
                    posDict[thisPos].append(motType)
    tempStr = ''
    if aMot:
        if '(' in aMot:
            tempStr = aMot[(aMot.index('(') +
                      1):aMot.index(')')]
        else:
            tempStr = aMot[0:aMot.index(' ')]
    fOut = open('posMotTable.' + prefix + tempStr + '.txt',
           'w')
    for i in motList:
        fOut.write('\t' + i)

    for i in posDict.keys():
        fOut.write('\n' + i)
        for j in motList:
            fOut.write('\t' + ('1' if (j in posDict[i]) else
              '0'))

def getBaseCounts():
    fRand = open('adjRandomPosListAll.txt')
    fOut = open('gctable.txt', 'w')

    posList = os.listdir('regrna_results')
    fOut.write('Sequence Hash\tSequence Type')
    randList = []

    header = fRand.readline()

    for line in fRand:
        lineList = line.split('\t')
        randList.append(lineList[0].strip() + '.' +
          lineList[1].strip() + '.' + lineList[2].strip())

    baseList = ['a', 'c', 't', 'g']
    diBaseList = []
    dummy = map(lambda x: diBaseList.extend(x), map(lambda y:
```

```
      [y + z for z in baseList], baseList))

   for i in baseList + diBaseList:
       fOut.write('\t' + i.upper() + ' Count')

   for i in posList:
       aPos = i.strip('.txt')
       thisPos = getPos(aPos)
       thisChr = thisPos[0]
       thisStart = thisPos[1]
       thisLen = thisPos[2]
       baseCounts = {}

       thisSeq = genomelib.get_sequence(thisChr,
                 int(thisStart), int(thisLen))
       lowerSeq = thisSeq.lower()
       for k in baseList:
           baseCounts[k] = len(filter(lambda x: x==k,
           lowerSeq))
       for k in diBaseList:
           baseCounts[k] = 0
       for j in range(len(lowerSeq)-1):
           if 'n' not in lowerSeq[j:j+2]:
               baseCounts[lowerSeq[j:j+2]] += 1

       fOut.write('\n' + aPos + '\t' + ('random' if aPos in
         randList else 'variant'))
       for k in baseList + diBaseList:
           fOut.write('\t' + str(baseCounts[k]))
   fOut.close()

def cleanFiles():
   fOpen = open('regionsCovered.txt')
   header = fOpen.readline()
   fileList = []
   for line in fOpen:
       lineList = line.split('\t')
       fileList.append(lineList[0].strip() + '.' +
        lineList[1].strip() + '.' + lineList[2].strip() +
        '.txt')
   fOpen.close()
   regrnaList = os.listdir('regrna_results')

   counter = 0

   for i in regrnaList:
       if i not in fileList:
```

```
            os.remove('regrna_results/' + i)
            counter += 1
    print(str(counter) + " files removed.")

def removeEmpty():
    fiList = os.listdir('regrna_results')
    for i in fiList:
        fOpen = open('regrna_results/' + i)
        header = fOpen.readline()
        tempList = []
        for line in fOpen:
            tempList.append(line.strip())
        if not tempList:
            shutil.move('regrna_results/' + i, 'emptyseqs')

def caseMotifs(numType='raw'):
    fMuts = open('posMotTable.mut.txt')
    header = fMuts.readline()
    headerList = header.strip().split('\t')

    caseN = 28
    caseDict = {}
    for i in range(1, caseN+1):
        tempNom = 'case' + str(i)
        caseDict[tempNom] = {}
        caseDict[tempNom]['muts'] = []
        for i in headerList:
            caseDict[tempNom][i] = 0
        caseDict[tempNom]['none'] = 0
        caseDict[tempNom]['tot'] = 0
        caseDict[tempNom]['totMuts'] = 0
        fOpen = open(os.path.expanduser('~/data/' + tempNom +
         '/processed/info/' + tempNom +
         '.regMuts.filtered.txt'))
        for line in fOpen:
            lineList = [x.strip() for x in line.split('\t')]
            tempHash = lineList[2] + lineList[5] + lineList[6]
                       +  lineList[3]
            if tempHash not in caseDict[tempNom]['muts']:
                caseDict[tempNom]['muts'].append(tempHash)
        fOpen.close()

    for line in fMuts:
        lineList = [x.strip() for x in line.split('\t')]
        thisHash = lineList[0]
        caseList = filter(lambda x: thisHash in
caseDict[x]['muts'],
```

```
                caseDict.keys())
        for j in caseList:
            caseDict[j]['totMuts'] += 1
        if all([x=='0' for x in lineList[1:]]):
            for j in caseList:
                caseDict[j]['none'] += 1
        else:
            for i in range(1, len(lineList)):
                aFlag = False
                if int(lineList[i]):
                    for j in caseList:
                        caseDict[j][headerList[i-1]] += 1
                        if not aFlag:
                            aFlag = True
                            caseDict[j]['tot'] += 1

    fMuts.close()

    fOut = open('caseMotInfo.txt', 'w')
    fOut.write(header.strip())
    fOut.write('\tnone\tTotal Mutations\n')
    for i in range(1, caseN+1):
        tempNom = 'case' + str(i)
        fOut.write(tempNom)
        for j in headerList:
            if numType == 'raw':
                fOut.write('\t' + str(caseDict[tempNom][j]))
            else:
                fOut.write('\t' +
                  str(float(caseDict[tempNom][j])/
                  caseDict[tempNom]['tot']))
        fOut.write('\t' + str(caseDict[tempNom]['none']) + '\t'
          + str(caseDict[tempNom]['totMuts']))
        fOut.write('\n')
    fOut.close()

def seqByType():
    fVars = open('poslist2.txt')
    fFake = open('fakePosList.reg.buffer.txt')
    fFake2 = open('fakeMutList4.txt')
    fFake3 = open('fakeMutList.fakeonly.reg.txt')

    listDict = {'vars':[], 'fake':[], 'fake2':[], 'fake3':[]}

    for line in fVars:
        listDict['vars'].append(line.strip())
    fVars.close()
```

```
    for line in fFake:
        listDict['fake'].append(line.strip())
    fFake.close()

    for line in fFake2:
        listDict['fake2'].append(line.strip())
    fFake2.close()

    for line in fFake3:
        listDict['fake3'].append(line.strip())
    fFake3.close()

    seqCount = {'vars':{'count':0, 'list':[]},
                'fake':{'count':0, 'list':[]},
                'fake2':{'count':0, 'list':[]},
                'fake3':{'count':0, 'list':[]}}

    for i in listDict.keys():
        for j in listDict[i]:
            if checkIfCovered(j):
                tempCov = getCover(j)
                if checkIfFileExists(tempCov) and tempCov not
                   in seqCount[i]['list']:

                    seqCount[i]['count'] += 1
                    seqCount[i]['list'].append(tempCov)

    for i in seqCount.keys():
        print i, seqCount[i]['count']


def getAllChrMotifs():
    global gloMotDict
    allChr = os.listdir('motifInfo')
    for i in allChr:
        gloMotDict[i.strip('.tx').strip('chr')] =
procChrMotifs(i)

gloChrDict = readRegions()
gloMotDict = {}
```

1.7 hgTables_slim.py

```
import sys, os, re
import random
from genomicode import genomelib
```

```
from run_regrna import getPos

def slimFiles():
   fOpen = open('hgTables.txt')
   geneDict = {}

   header = fOpen.readline()
   for line in fOpen:
       if line.strip():
           lineList = line.split('\t')
           thisGene = lineList[12]
           if thisGene not in geneDict:
               geneDict[thisGene] = {'start':
                 int(lineList[4]), 'end':int(lineList[5]),
                 'chr':lineList[2]}
           else:
               geneDict[thisGene]['start'] =
                min(geneDict[thisGene]['start'],
               int(lineList[4]))
               geneDict[thisGene]['end'] =
                max(geneDict[thisGene]['end'],
          int(lineList[5]))
   fOpen.close()

   fOut = open('hgTables.slim.txt', 'w')
   fOut.write('Gene\tChromosome\tStart\tEnd')
   for i in geneDict.keys():
       fOut.write('\n' + i + '\t' + geneDict[i]['chr'] +
          '\t' + str(geneDict[i]['start']) + '\t' +
          str(geneDict[i]['end']))
   fOut.close()

def orgSegs():
   fOpen = open('hgTables.slim.txt')
   chrDict = {}
   header = fOpen.readline()
   for line in fOpen:
       lineList = line.split('\t')
       thisChr = lineList[1]
       if thisChr not in chrDict:
           chrDict[thisChr] = []
       posStart = int(lineList[2])-25000
       if posStart < 0:
           posStart = 0
       posEnd = int(lineList[3])+25000
       chrDict[thisChr].append((posStart, posEnd))
   fOpen.close()
```

```
    chrList = ['chr' + str(x) for x in range(1, 23)]
    chrList.append('chrX')
    chrList = filter(lambda x: x in chrDict.keys(), chrList)

    sortChrDict = {}

    for i in chrList:
        posList = chrDict[i]
        posList.sort(key=lambda x: x[0])
        tempList = []
        tempTup = (0,0)
        for j in posList:
            if (j[0] <= tempTup[1] and j[0] >= tempTup[0]) or
               (j[1] <= tempTup[1] and j[1] >= tempTup[0]):
                tempTup = (min(tempTup[0], j[0]),
                           max(tempTup[1], j[1]))
            else:
                if not tempTup==(0,0):
                    tempList.append(tempTup)
                tempTup=j
        if not tempList and not tempTup==(0,0):
            tempList.append(tempTup)
        sortChrDict[i] = tempList
    return sortChrDict

def pickRan(aNum=1000):
    chrDict = orgSegs()
    lenDict = {}
    proDict = {}
    for i in chrDict.keys():
        lenDict[i] = [x[1]-x[0]+1 for x in chrDict[i]]
        tempList = [float(x)/sum(lenDict[i]) for x in
                    lenDict[i]]
        proDict[i] = [sum(tempList[0:x]) for x in range(1,
                      len(tempList)+1)]
    randChr = [random.choice(chrDict.keys()) for x in
               range(aNum)]
    posList = []
    while len(posList) < aNum:
        i = randChr[len(posList)]
        thisList = proDict[i]
        ranNum = random.random()
        temp = filter(lambda x: x >= ranNum, thisList)
        thisRange = chrDict[i][0]
        thisPos = random.randint(thisRange[0], thisRange[1])
        tempHash = i.strip('chr') + 'NN' + str(thisPos)
```

```
        if tempHash not in posList and thisPos > 200 and not
           genomelib.get_sequence(i.strip('chr'), thisPos,
           1).lower()=='n':

            posList.append(tempHash)

    fOut = open('fakePosList4.txt', 'w')
    for i in posList:
        fOut.write(i + '\n')

def getExons():
    fOpen = open('hgTables.txt')
    header = fOpen.readline()
    chrDict = {}
    for line in fOpen:
        if line:
            lineList = line.split('\t')
            thisChr = lineList[2].strip('chr')
            if thisChr not in chrDict:
                chrDict[thisChr] = []
            exStarts = lineList[9].strip(',').split(',')
            exEnds = lineList[10].strip(',').split(',')
            for i in range(len(exStarts)):
                chrDict[thisChr].append((int(exStarts[i]),
                 int(exEnds[i])))
    fOpen.close()
    return chrDict

def cleanExon():
    chrDict = getExons()
    fOpen = open('fakePosList4.txt')
    posList = []
    for line in fOpen:
        posList.append(line)
    fOpen.close()
    fOut = open('fakePosList4.reg.txt', 'w')
    for i in posList:
        thisPos = getPos(i)
        thisChr = thisPos[0]
        thisBase = int(thisPos[1])

        if thisChr in chrDict:
            theseEx = chrDict[thisChr]
            if not any([x[0] <= thisBase and thisBase<=x[1] for
                x in theseEx]):
                fOut.write(i)
    fOut.close()
```

1.8 regression2.R

```
setwd("H:/Documents/Actual documents")
library(gplots)
library(survival)
caseInfo <- read.table('caseMotInfo.txt', quote='', header=T,
sep='\t')
caseInfo.order <- caseInfo[order(caseInfo[,1]),]

####
keggInfo <- read.table('allPathsScores.dled.txt', quote='',
header=T, sep='\t')
keggInfo.order <- keggInfo[order(keggInfo[,1]),]

keggCoding <- read.table('allPaths_codMuts.txt', quote='',
header=T, sep='\t')
keggCoding.order <- keggCoding[order(keggCoding[,1]),]
###
pathList = colnames(keggInfo)[-1:-2]
outList = list()

for(aPath in pathList){

     caseInfo.temp <- cbind(caseInfo.order,
     keggCoding[,aPath])
     colnames(caseInfo.temp) <- c(colnames(caseInfo.order),
     'Coding.Muts')
     linMat <-cbind(keggInfo[,aPath],
     sweep(caseInfo.temp[,c(2:20,22)],1,(caseInfo.temp$Total.M
     utations+caseInfo.temp$Coding.Muts),'/'))
     colnames(linMat) <- c('Score', 'ncRNA', 'longStems',
       'funcRNA', 'miRNA', 'Transcript', 'RIT', 'PAS', 'ISE',
       'ORF', 'ESE', 'ERPIN', 'ESS', 'UTR', 'ARE', 'Splicing',
       'RBS', 'ISS', 'C2U', 'none', 'Coding')
     rownames(linMat) <- caseInfo.temp[,1]

     tempList = list()
     kccModel <- lm(Score ~
      0+ncRNA+longStems+funcRNA+miRNA+Transcript+RIT+PAS+ISE+
      ORF+ESE+ERPIN+ESS+UTR+ARE+Splicing+RBS+ISS+C2U,
     data=linMat)
     kccModel.code <- lm(Score ~ 0+Coding, data=linMat)
     kccModel.all <- lm(Score ~
      0+ncRNA+longStems+funcRNA+miRNA+Transcript+RIT+PAS+ISE+
      ORF+ESE+ERPIN+ESS+UTR+ARE+Splicing+RBS+ISS+C2U+Coding,
      data=linMat)
```

```
        kccModel.sig <- lm(Score ~
         0+ncRNA+Splicing+longStems+ORF+RIT+miRNA+funcRNA+
         Transcript+ESE+ESS+ISE+Coding, data=linMat)

        tempList[['NC']] <- summary(kccModel)
        tempList[['Code']] <- summary(kccModel.code)
        tempList[['All']] <- summary(kccModel.all)
        tempList[['Sig']] <- summary(kccModel.sig)

        outList[[aPath]] = tempList
}
```

BIBLIOGRAPHY

American Cancer Society. Cancer Facts & Figures 2016. Atlanta: American Cancer Society; 2016.

Bailey SD, Desai K, Kron KJ, Parisa M, Sinnott-Armstrong NA, Treloar AE, Dowar M, Thu KL, Cescon DW, Silvester J, Yang SYC, Wu X, Pezo RC, Haibe-Kains B, Mak TW, Bedard PL, Pugh TJ, Sallari RC, Lupien M. Noncoding somatic and inherited single-nucleotide variants converge to promote ESR1 expression in breast cancer. Nat Genet. 2016;48(10):1260-6.

Balin SJ, Cascalho M. The rate of mutation of a single gene. *Nucleic Acids Res*. 2010;38(5):1575-82.

Broad Institute TCGA Genome Data Analysis Center (2016): Analysis-ready standardized TCGA data from Broad GDAC Firehose 2016_01_28 run. Broad Institute of MIT and Harvard. Dataset. https://doi.org/10.7908/C11G0KM9

Chang JT, Gatza ML, Lucas JE, Barry WT, Vaughn P, Nevins JR. SIGNATURE: a workbench for gene expression signature analysis. BMC Bioinformatics. 2011;12:443.

Chang TH, Huang HY, Hsu JB, Weng SL, Horng JT, Huang HD: An enhanced computational platform for investigating the roles of regulatory RNA and for identifying functional RNA motifs. BMC bioinformatics 2013, 14 Suppl 2:S4

Chen X, Chang JT. Planning bioinformatics workflows using an expert system. Bioinformatics. 2017;33(8):1210-1215.

Cibulskis K, Lawrence MS, Carter SL, Sivachenko A, Jaffe D, Sougnez C, Gabriel S, Meyerson M, Lancer ES, Getz G. Sensitive detection of somatic point mutations in impure and heterogeneous cancer samples. *Nat Biotechnol*. 2013;31(3):213-9.

Comprehensive molecular profiling of lung adenocarcinoma*. Nature*. 2014;511(7511):543-50.

Depristo MA, Banks E, Poplin R, Garimella KV, Maguire JR, Hartl C, Philippakis AA, del Angel G, Rivas MA, Hanna M, McKenna A, Fennell TJ, Kernytsky AM, Sivachenko AY, Cibulskis K, Gabriel SB, Altshuler D, Daly MJ. A framework for variation discovery and genotyping using next-generation DNA sequencing data. Nat Genet. 2011;43(5):491-8.

Elgar G, Vavouri T. Tuning in to the signals: noncoding sequence conservation in vertebrate genomes. *Trends Genet.* 2008;24(7):344-52.

Fredriksson NJ, Ny L, Nilsson JA, Larsson E. Systematic analysis of noncoding somatic mutations and gene expression alterations across 14 tumor types*. Nat Genet*. 2014;46(12):1258-63.

Hartwell LH, Kastan MB (1994). Cell cycle control and cancer. Science-AAAS-Weekly Paper Edition, 266(5192), 1821-1828.

Horn S, Figl A, Rachakonda PS, Fischer C, Sucker A, Gast A, Kadel S, Moll I, Nagore E, Hemminki K, Schadendorf D, Kumar R. TERT promoter mutations in familial and sporadic melanoma. *Science*. 2013;339(6122):959-61.

Huang FW, Hodis E, Xu MJ, Kryukov GV, Chin L, Garraway LA. Highly recurrent TERT promoter mutations in human melanoma. *Science*. 2013;339(6122):957-9.

Kanehisa M, Goto S. KEGG: Kyoto encyclopedia of genes and genomes. Nucleic Acids Res. 2000;28(1):27-30.

Kanehisa M, Sato Y, Kawashima M, Furumichi M, Tanabe M. KEGG as a reference resource for gene and protein annotation. Nucleic Acids Res. 2016;44(D1):D457-62.

Kanehisa M, Furumichi M, Tanabe M, Sato Y, Morishima K. KEGG: new perspectives on genomes, pathways, diseases and drugs. Nucleic Acids Res. 2017;45(D1):D353-D361.

Lee W, Jiang Z, Liu J, Haverty PM, Guan Y, Stinson J, Yue P, Zhang Y, Pant KP, Bhatt D, Ha C, Johnson S, Kennemer MI, Mohan S, Nazarenko I, Watanabe C, Sparks AB, Shames DS, Gentleman R, de Sauvage FJ, Stern H, Pandita A, Ballinger DG, Drmanac R, Modrusan Z, Seshagiri S, Zhang Z. The mutation spectrum revealed by paired genome sequences from a lung cancer patient. *Nature*. 2010;465(7297):473-7.

Li H. (2013) Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. arXiv:1303.3997v1 [q-bio.GN].

Pao W, Girard N. New driver mutations in non-small-cell lung cancer. *Lancet Oncol.* 2011;12(2):175-80.

Pon JR, Marra MA. Driver and passenger mutations in cancer. Annu Rev Pathol. 2015;10:25-50.

Rheinbay E, Parasuraman P, Grimsby J, Tiao G, Engreitz JM, Kim J, Lawrence MS, Taylor-Weiner A, Rodriguez-Cuevas S, Rosenberg M, Hess J, Stewart C, Maruvka YE, Stojanov P, Cortes ML, Seepo S, Cibulskis C, Tracy A, Pugh TJ, Lee J, Zheng Z, Ellisen LW, Iafrate AJ, Boehm JS, Gabriel SB, Meyerson M, Golub TR, Baselga J, Hidalgo-Miranda A, Shioda T, Bernards A, Lander ES, Getz G. Recurrent and functional regulatory mutations in breast cancer. Nature. 2017;547(7661):55-60.

Tomasetti C, Vogelstein B. Cancer etiology. Variation in cancer risk among tissues can be explained by the number of stem cell divisions. *Science*. 2015;347(6217):78-81.

Wang K, Li M, Hakonarson H. ANNOVAR: functional annotation of genetic variants from high-throughput sequencing data. *Nucleic Acids Res.* 2010;38(16):e164.

Weinhold N, Jacobsen A, Schultz N, Sander C, Lee W. Genome-wide analysis of noncoding regulatory mutations in cancer. Nat Genet. 2014;46(11):1160-5.

Weinstein JN, Collisson EA, Mills GB, Shaw KR, Ozenberger BA, Ellrott K, Shmulevich I, Sander C, Stuart JM. *The Cancer Genome Atlas Pan-Cancer analysis project. Nat Genet.* 2013;45(10):1113-20.

Wittkopp PJ, Kalay G. Cis-regulatory elements: molecular mechanisms and evolutionary processes underlying divergence. *Nat Rev Genet*. 2012;13(1):59-69.

Wood LD, Parsons DW, Jones S, Lin J, Sjöblom T, Leary RJ, Shen D, Boca SM, Barber T, Ptak J, Silliman N, Szabo S, Dezso Z, Ustyanksky V,Nikolskaya T, Nikolsky Y, Karchin R, Wilson PA, Kaminker JS, Zhang Z, Croshaw R, Willis J, Dawson D, Shipitsin M, Willson JK, Sukumar S,Polyak K, Park BH, Pethiyagoda CL, Pant PV, Ballinger DG, Sparks AB, Hartigan J, Smith DR, Suh E, Papadopoulos N, Buckhaults P, Markowitz SD, Parmigiani G, Kinzler KW, Velculescu VE, Vogelstein B. The genomic landscapes of human breast and colorectal cancers. Science. 2007;318(5853):1108-13.

VITA

Kevin Wen Zhu was born in Houston, Texas on March 30, 1991, the son of Dr. Michael Zhu and Dina Chuang-Zhu. After completing high school at Worthington Kilbourne High School, Columbus, Ohio in 2009, he entered Rice University in Houston, Texas. He received the degree of Bachelor of Arts with a major in Computational and Applied Mathematics from Rice in May, 2013. For the next year, he worked as a research assistant at the University of Texas Health Science Center, School of Biomedical Informatics. In August of 2014, he entered the University of Texas Graduate School of Biomedical Sciences.

Permanent address:

2828 W Holcombe Blvd. #K

Houston, TX 77025