2019-01-01

A Computational Method For Predicting Functional Effects Of Cancer-Related Genetic Sequence Variants

Bofei None Wang
University of Texas at El Paso, bwang2@miners.utep.edu

A COMPUTATIONAL METHOD FOR PREDICTING FUNCTIONAL EFFECTS OF CANCER-RELATED GENETIC SEQUENCE VARIANTS

BOFEI WANG

Master's Program in Computational Science

APPROVED:

Ming-Ying Leung, Ph.D., Chair

Marc B. Cox, Ph.D.

M. Shahriar Hossain, Ph.D.

Sangjin Kim, Ph.D.

Steven Crites, Ph.D.
Dean of the Graduate School

A COMPUTATIONAL METHOD FOR PREDICTING FUNCTIONAL EFFECTS OF CANCER-RELATED GENETIC SEQUENCE VARIANTS

by

BOFEI WANG

THESIS

Presented to the Faculty of the Graduate School of

The University of Texas at El Paso

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE

Computational Science Program

THE UNIVERSITY OF TEXAS AT EL PASO

May 2019

Acknowledgements

I would like to express my sincere gratitude to my advisor, Dr. Ming-Ying Leung, the director of Computational Science Program at The University of Texas at El Paso, for her patience, encouragement, motivation, immense advice and constant support. She always provides insightful suggestions and clear explanations when I was lost. She put a lot efforts helping me improve my writing ability. When I feel tired, she constantly driving me with energy. I can never reach this point without enormous help from Dr. Leung.

I also want to thank other members in my committee, Dr. Hossain of Computer Science Department, Dr. Cox of Biological Sciences Department and Dr. Kim of Mathematical Sciences Department, all at The University of Texas at El Paso. Their comments and additional guidance were invaluable to my project and even to my career. My heartfelt gratitude also goes to the research group members: Jonathan Mohl, David Armendariz, Angelica Marquez and Mariana Marquez for their enormous participation in putting together this work. Additionally, a special gratitude goes to funding sources NIH Grant 5G12RR007592.

Finally, I want to thank my parents. They gave me huge support for going abroad, studying and living here. During the completion of this work, they supported me spiritually when I was frustrated by difficulties. I have no words to express my love.

Abstract

Rapid advances in next generation sequencing (NGS) technologies provide many opportunities to identify associations between genetic sequence variants (GSV) and diseases, which may lead to better clinical diagnosis and treatments. OncoMiner is a bioinformatics pipeline developed at UTEP (OncoMiner.utep.edu) for mining NGS data. It can identify exonic sequence variants, link them with associated literatures, visualize genomic locations and compare their occurrence frequencies among different groups. However, the current version of OncoMiner is limited to accepting only a specific input file format provided by the Otogenetics NGS Lab Services. The main objectives of my current work are (1) to develop a Python script for preprocessing the more widely used variant calling format (VCF) NGS files and convert them to the OncoMiner input (OMI) format, and (2) to evaluate the performance of the script.

Most of the required data fields in the OMI file can be extracted directly from the VCF file. The genomic region type, however, needs to be determined by comparison with a reference genome. Since I will be working on human cancer data, the reference genome used for this work is the human genome assembly hg38 obtained from UCSC Genome Browser. To improve efficiency, the script splits the VCF file and reference genome by chromosomes into smaller files for parallel processing. Our script has been tested on 148 VCF files, containing data from prostate cancer patients, downloaded from The Cancer Genome Atlas (TCGA). Parallelization of the script obtained average speedups of 1.50, 2.28, 3.14, 3.84, 4.00 using 2, 4, 8, 16, 24 cores respectively. To test the programs capability of handling big datasets, 35 larger files with sizes ranging from 193.8 MB to 3.7 GB are used. These files contain data from leukemia patients, cell lines, and normal individuals collected at local hospitals and UTEP. Both the number of variants and the number of samples in the VCF file were found significantly correlated with runtime. A multiple linear regression indicated that 83% variation in the runtime can be explained by its relationship with the numbers

of variants and samples.

We plan to incorporate this preprocessing script into OncoMiner pipeline and use it for downstream analyses of a collection of 500 prostate cancer VCF files from TCGA, and the local leukemia dataset to identify GSVs associated with the diseases and prioritize risky variants based on their predicted functional effects.

Table of Contents

Appendix

List of Tables

List of Figures

Chapter 1

Introduction

High-throughput sequencing technologies have become indispensable tools for genomics, transcriptomics and proteomics studies. Since the beginning of the 21st century, more and more biotechnology companies started to release next generation DNA sequencing platforms and enormous amounts of data have been generated by sequencing millions of DNA molecules at the same time. Computational tools and data analytics techniques designed for handling and mining these next generation sequencing (NGS) data have helped provide useful information and accelerated the research in many fields of biological sciences. In 2016, our research group has initiated the implementation of a bioinformatics pipeline called OncoMiner (OncoMiner.utep.edu) to facilitate analysis of exonic sequence variants. In the past few years, we have continued to generalize and expand the functionalities of OncoMiner with the ultimate goal of establishing a reliable and versatile computational tool for mining NGS data to identify cancer related genetic sequence variants (GSVs) and predict their functional effects, which could, in turn, help discover potential biomarkers for cancer diagnosis.

My project focuses on developing additional OncoMiner modules for processing and analyzing two particular datasets of interests to UTEP researchers. The first set consists of sequence data of targeted genes from 500 patients with prostate cancer (PrCa). The second set consists of whole exome sequence (WES) data obtained from leukemia patients, normal individuals, and cell lines in local hospitals and UTEP.

1.1 Background and Significance

Deoxyribonucleic acid (DNA) is a macromolecule that carries genetic information of individuals. It consists of two chains that coil against each other, forming a double helix structure. The basic unit of DNA is the nucleotide, which is composed of a deoxyribose, one of four nitrogenous bases, and a phosphate group. The variation in nitrogenous bases (cytosine, guanine, adenine, thymine) makes people different from each other. However, mutations in these bases may also cause changes to certain genes and result in diseases. Many cancers are linked to genetic mutations in the human genome[1]. which contains around 3 billion base pairs. Thus, identifying GSVs in the human genome can play an important role in the detection and treatment of cancers. Yet, the cost of sequencing an entire human genome accurately is still rather formidable. Instead, many researchers prefer use a more cost-effective strategy by sequencing only certain targeted genes of interest, or the whole exome which comprises a small percentage of the genome containing sequences responsible for encoding proteins. GSVs occurring on these select regions are expected to be more likely to be disease related.

DNA changes can occur in a small scale as point mutations, or in a larger scale that could result in altering the structure of chromosomes (e.g., copy number variations, chromosome translocations). My work focuses only on point mutations. There are basically three types of DNA point mutation, substitution, insertion and deletion. Substitution occurs when a nucleotide is substituted for another. Insertion occurs when extra nucleotides are inserted into the DNA sequence. Deletion occurs when nucleotides are removed from the sequence. An example of three types of point mutation are shown in Figure 1.1. Identification and annotation of these GSVs require a standardized workflow which will be described in detail in chapter 2.

Discovering cancer related GSVs and understanding their functional effects are expected to play increasingly important roles in cancer diagnosis, treatment, and prognosis. As the huge amounts of variations in the genome make it hard to experimentally detect cancer

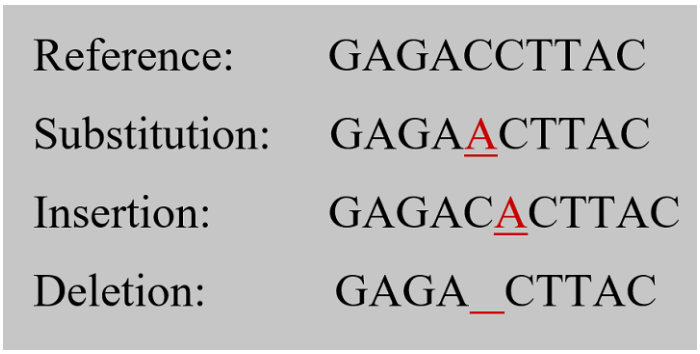

Figure 1.1: Three types of point mutation in DNA

associated GSVs, efficient computer programs to identify those GSVs that are likely to be associated with cancer, and predicting their possible functional effects have become essential tools for biomedical researchers. OncoMiner is one of the bioinformatics pipelines set up for such purposes.

PrCa is a common disease in men. According to American cancer society, there will be 174650 estimated new prostate cancer cases in 2019[2]. As it is widely acknowledged that population-based prostate-specific antigen (PSA) screening causes overdiagnosis of low risk cancer[3], the demand for a better understanding of the genetic causes of PrCa is urgent. Although over 163 PrCa risk variants have been identified in different studies[4], their diagnosis and prognosis relevance remains unclear. Further investigations to identify and interpret more common variants in PrCa can have great potential to improve precision diagnosis, and risk prediction.

Leukemia is cancer of the blood cells, usually affecting the white blood cells. There are four main types of leukemia: acute lymphocytic leukemia (ALL), acute myeloid leukemia (AML), chronic myeloid leukemia (CML), and chronic lymphocytic leukemia (CLL). Particularly, AML and ALL are drawing more attention as they progress rapidly without proper treatment. Despite drastic improvement in cure rates, relapse still occurs in approximately 15% of ALL patients[5]. Prognosis remains poor for AML patients, with less than 30% of patients surviving within a year of diagnosis[6]. Identifying related GSVs in AML and ALL

is necessary and may provide theoretical basis for clinical treatment and prognosis.

1.2 OncoMiner Pipeline

A number of software packages have been developed by various research groups for NGS data analysis. OncoMiner[7] is a bioinformatics pipeline developed at UTEP initially for mining a local set of WES data. It was implemented in 2016 as a web server (OncoMiner.utep.edu) to assist biomedical researchers in our Border Biomedical Research Center (BBRC) at UTEP to analyze the datasets obtained from the local population and cell lines. The pipeline can identify GSVs, link them with associated research literature, visualize their genomic locations, and compare their occurrence frequencies among different groups of subjects.

The OncoMiner homepage is shown in Figure 1.2. Users can submit a job by uploading an input data file and entering an email address. Once the processing is done, the user will be notified by an email containing a link to the annotated output file. OncoMiner is also able to generate Cricos diagrams and conduct statistical comparisons for multiple output files.

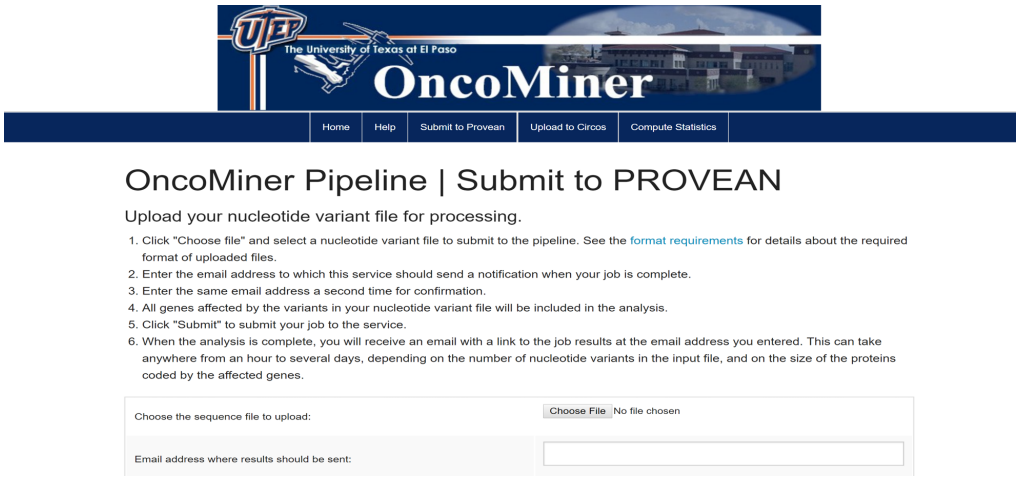

Figure 1.2: An overview of Oncominer pipeline

Because the original design of the current OncoMiner pipeline is centered around the local datasets, its flexibility to handle different types of NGS datasets in general has not been fully developed. In particular, the OncoMiner input (OMI) file is limited to a specific file format set by Otogenetics company, which requires 11 fields including a chromosome number, location of GSVs, reference base, mutated base, a quality score, the genomic region, etc. An example of OMI file is shown in Figure 1.3.

var_index	chrom	gene_name	left	right	ref_seq	var_seq1	var_seq2	count1	count2	var_score	where_in_transcript	change_type1
1	chr1	FAM213B	2587116	2587117	G	G	T	10	3	28	Not translated, ncRNA	
1	chr1	FAM213B	2587116	2587117	G	G	T	10	3	28	CDS	Non-synonymous
2	chr1	PADI2	17071327	17071328	G	G	T	18	2	28	Intron	
3	chr1	NBPF3	21436588	21436589	G	G	T	12	3	28	5' untranscribed	
4	chr1	HSPG2	21908471	21908472	G	G	T	7	2	28	Intron	
4	chr1	HSPG2	21908471	21908472	G	G	T	7	2	28	Intron	
5	chr1	MYOM3	24097770	24097771	G	G	T	32	3	28	Intron	
6	chr1	GPATCH3	26900427	26900428	C	A	C	3	51	28	CDS	Synonymous
7	chr1	SLC9A1	27101342	27101343	C	A	C	2	11	28	Intron	

Figure 1.3: Oncominer Input file example

In order to use OncoMiner on more general datasets, we need to develop a preprocessing program to handle the more common NGS files in FastQ, BAM/SAM and VCF formats and convert them to the OMI format for downstream analysis. The codes for preprocessing FastQ and BAM/SAM files in parallel environments have already been developed by other members of our group[8]. I will complete the OncoMiner preprocessing (OP) program by developing the functionality to preprocess VCF (variant calling format) files. An example of VCF file is shown in Figure 1.4

##fileformat=VCFv4.2
##fileDate=20160528
##center="NCI Genomic Data Commons (GDC)"

#CHROM	POS	ID	REF	ALT	QUAL	FILTER	INFO	FORMAT	NORMAL	TUMOR
chr1	1657358	rs3772302	T	TA	.	alt_allele_i	DB;ECNT=1	GT:AD:AF:	0/0:617,69	0/1:437,32
chr1	1657665	.	G	A	.	germline_r	ECNT=1;HC	GT:AD:AF:	0/0:14,0:0.	0/1:3,2:0.4
chr1	1738993	rs7707514	T	C	.	alt_allele_i	DB;ECNT=1	GT:AD:AF:	0/0:631,25	0/1:550,17

Figure 1.4: Variant calling format file example

1.3 Specific Aims

The specific aims of my M.S. thesis work are:

1. Develop a Python code to efficiently preprocess NGS data files in VCF format and convert them to OMI files that can be inputted to OncoMiner for further analysis. Since some of the data files are very large, the runtime is expected to be long. I will design this code to run with multiprocessing on parallel environments.

2. Evaluate the performance of the code developed in Aim 1, and the parallelization efficiency using VCF files of various sizes. I will analyze the runtime behavior of the program with large VCF data files and explore its correlations with input file size, number of variants, and number of samples.

The work completed for the above aims will be the foundation for my Ph.D. research, where the goal is to identify a list of candidate GSVs that are likely to be involved in PrCa or leukemia. A more detailed description of my Ph.D. project plan will be given at the end of this thesis.

Chapter 2

Literature Review

In this chapter, I provide some basic information about Acute Myeloid Leukemia and whole genome sequence (WGS), along with the developed software tools for WGS analysis. At the end, I also include some brief introduction to related computation techniques and statistical tools. Material of DNA sequencing and popular programs for WGS analysis can also be found in basic textbooks.

2.1 Prostate Cancer

Prostate cancer (PrCa) is one of the most common cancers and is among the leading causes of cancer-related mortality among men. It alone accounts for almost 20% of newly diagnosed cancer patients in 2018[9]. It occurs in the prostate, a gland in men that produces the seminal fluid that nourishes and transports sperm. The detection is usually based on the abnormal prostate-specific antigen because of the lack of obvious symptoms. Age is closely related to the risk of PrCa. It is rare among men younger than 40 years of age. But the incidence rate of PrCa increases dramatically after 40 years of age. Only 1 in 10,000 under age 40 will be diagnosed with PrCa, which is low compared to 1 in 39 for ages $40-59$ and 1 in 14 for ages $60-69$[10].

High throughput technology has enabled us to identify the differences in genome between individuals. These variants between normal people and PrCa patients may be the cause of disease. Since 2007, many PrCa risk-associated SNPs have been identified through genome-wide association studies (GWAS). Till 2018, more than 163 SNPs have been confirmed to be associated with the risk of PrCa[4]. The effect of a single variant is usually modest, so

many studies focus on the cumulative risk of a number of genetic variants. Zheng et al evaluated 16 SNPs from five chromosomal regions (three at 8q24 and one each at 17q12 and 17q24.3) in a Swedish population. They assessed the individual and combined association of the SNPs with PrCa. Five SNPs were found to be weakly associated with PrCa, but their combined effect was much stronger[11]. Apart from these five variants, Helfand et al also identified 4 additional variants located on on chromosomes 2p15, 10q11, 11q13 and Xp11[12]. They assessed both the independent and cumulative risk of all 9 variants. Results suggested that PrCa cases had an increased incidence of all 9 risk variants compared to controls. A cumulative model with all 9 SNPs had greater prostate cancer risk stratification than a model with the previous 5 SNPs. Particularly, they found that men with 6 or more variants were at greater than 6-fold increased risk for PrCa. The identification of more and more variants could provide a better understanding of PrCa risk and also help with the prediction of incidence and prognosis.

2.2 Acute Myeloid Leukemia and Acute Lymphocytic Leukemia

Acute Myeloid Leukemia (AML) is a fast-growing form of cancer of the blood and bone marrow and is the most common type of acute leukemia in adults. It accounts for 1.3% of new cancer cases in the USA and affects about 0.5% of the population during their lifetime. Although AML can occur at any age stage, its incidence increases with age, with a median age at diagnosis of 68 years[13]. Leukemia starts in the tissues that forms blood. Most blood cells develop from cells in the bone marrow called stem cells. Advances in the exploration of stem cells and genomics discovery have uncovered the mechanism of how AML develops. The pathogenesis of AML involves genetic changes in hematopoietic stem cells and leads to the abnormal proliferation and differentiation of myeloid cells, resulting in the accumulation of a large number of immature myeloid cells[6].

The advent of high-throughput sequencing, also known as next generation sequencing (NGS), has led us to a stage where the genetic landscape of more and more cancers are defined. Many recent studies have revealed relevant genomic landscape and driver mutations of AML. The Cancer Genome Atlas Project (TCGA) analyzed the genomes of 200 clinically annotated adult cases of de novo AML using either whole-genome sequencing (50 cases) or whole-exome sequencing (150 cases), along with RNA and microRNA sequencing and DNA-methylation analysis. They identified an average of 13 coding mutations in genes, which is relatively fewer compared with other adult cancers. Among these mutations, at least one potential driver mutation was found in nearly all AML samples[14].Papaemmanuil et al in 2016 identified 5234 driver mutations involving 76 genes or regions in 1540 patients. In accord with the result from TCGA research, at least 1 driver mutation in 1478 of 1540 samples (96%), and 2 or more driver mutations in 86% of samples[15] were identified.

Acute lymphocytic leukemia (ALL) is another type of blood cancer and is the most prevalent cancer among children and adolescents in the United States, accounting for 20% of all cancers diagnosed in persons under 20 years old[16]. The American Cancer Society estimated that about 5930 new cases will appear in U.S. in 2019, with slightly more in men than in women. T-cell ALL (T-ALL) and B-cell precursor ALL(B-ALL) are two subtypes of the disease. About 50% of pediatric cases of B-ALL have either a translocation between chromosome 12 and 21 or hyperdiploidy. In about 60% of pediatric T-ALL, an overexpression of the transcription factor TAL1 is observed[17]. Several studies have identified genetic variants located in coding region and microRNAs that are associated with ALL [18, 19]. These discoveries may provide more insights into the classification, diagnosis and treatment of AML and ALL.

2.3 Whole Genome Sequencing Analysis

Human genome contains around 3 billion base pairs. Knowledge of DNA sequences plays a key role in biological and medical discovery. Traditional DNA sequencing methods,

like Sanger sequencing[20], are very time consuming and costly. Since 21 century, with the advent and development of rapid DNA sequencing technologies, a large amount of sequence data is able to be generated efficiently and with low cost. One of the most famous commerical platforms for next generation sequencing is called Genome Analyser, which was released by Illumina in 2006. It was able to sequence 1Gb of data in a single run. A series of more recent sequencers produced by Illumina and other companies were able to yield data about 30x human genome in less than 30 hours [21], or even more efficient.

Exome is the protein-coding region of the genome. It only constitutes 1-2% of the human whole genome, but about 85% of known disease-related mutations were found within this region[22]. So whole exome sequencing (WES) and the analysis and interpretation of these data could be cost-effective strategies to identify associations between GSV and diseases, which may lead to better clinical diagnosis and treatments. In this section, I will describe basic steps for analyzing WGS and/or WES data from these platforms. These steps and popular softwares for each step are summarized in Figure 2.1.

Fastq is a standard format for storing biological sequence data and is most commomly used. In general, all raw data from Illumina platforms is in fastq. Othr platforms also generate standard flowgram format (SFF), binary alignment map (BAM) and so on. Fastq is a text-based file using four lines for each sequence. It usually starts with a '@' character followed by a sequence identifier. The basic structure is the same as Fasta file, but fastq contains one line specifing the quality score. Since the data is unaligned raw data, the pre-processing step includes quality control (QC), alignment and post processing, like removal of low coverage and duplicate reads [23].

2.3.1 Quality Assessment

The first step after completing the sequencing run is to assess the quality of raw data and remove reads that do not meet pre-defined criterions. This early identification may reduce the amount of further downstream analyses. The quality of sequences is measured by a Phred score $Q_{PHRED} = -10 \times \log_{10}(P_e)$, where P_e is the estimated probability of error[24].

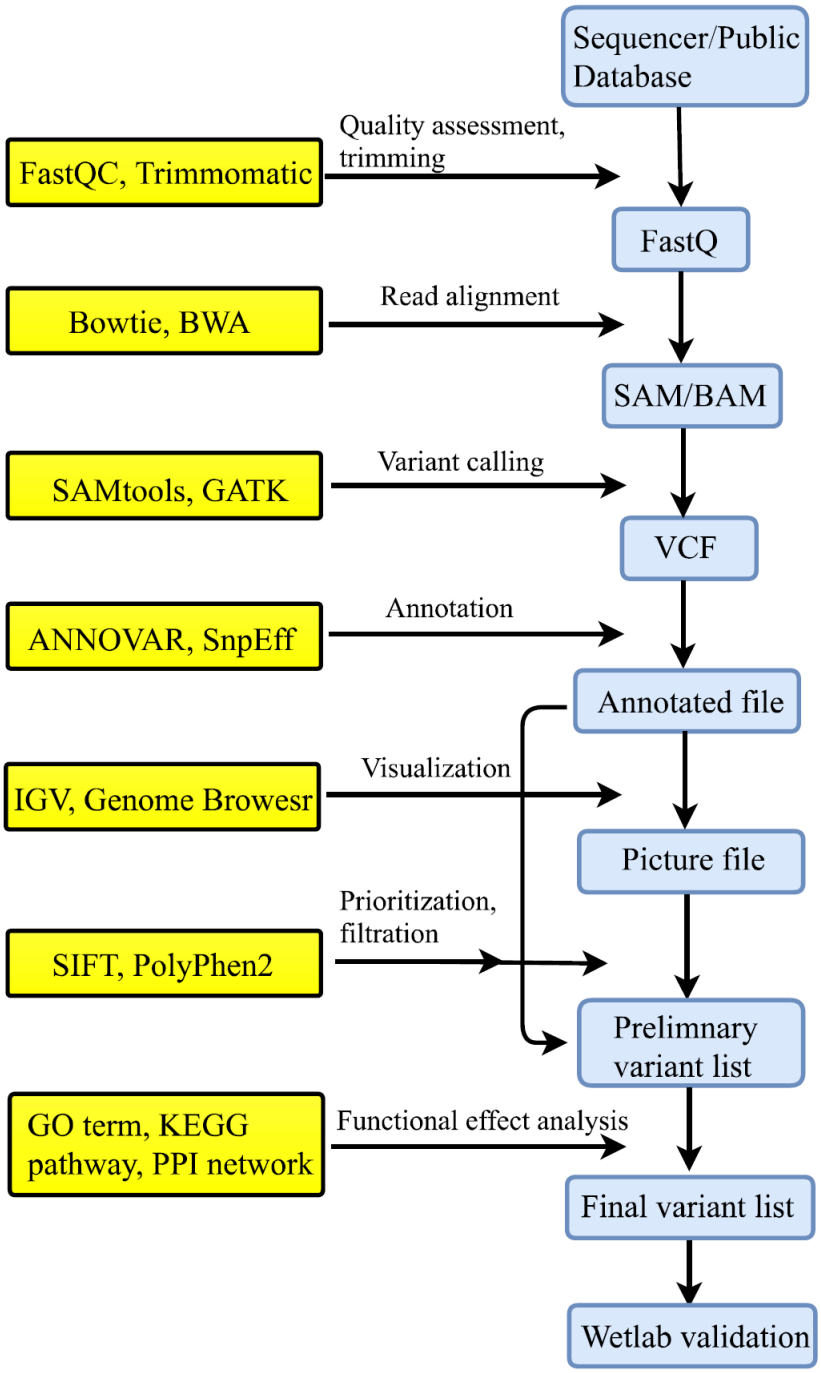

Figure 2.1: General steps for NGS analysis

Bases with a Phred score less than 20 are usually filtered out, although the criterion may be modulated by scientists according to their researches preference.The fastq file already incorporates the quality score. There are also several tools developed to assess the quality of raw data from NGS.Trimming low quality ends from reads and removal of contamination reads are also important as the alignment of contaminating sequences to a reference genome will result in low alignment quality.

2.3.2 Read alignment

After quality control and preprocessing, the next step is to align reads to a reference genome. This mapping is a classical string match, but due to the sheer amount of NGS data, this process involves complicated and computationally intensive work. Two main sources for the human reference genome assembly are the University of Santa Cruz (UCSC) and the Genome Reference Consortium (GRC). Two human assemblies are basicly the same but differ in their nomenclature[25]. Alignment results are usually stored in a Sequence Alignment Map (SAM) or its compressed binary form BAM.

Some bad features of sequences only show up after alignment, which makes post-processing important. The PCR amplification, which is performed to ensure good coverage and depth, may introduce duplicate reads to sequence data[26]. These duplicates can affect further downstream analyses. As a result, it is necessary to remove them before variant calling. Local realignment of insertions and deletions (indels) are also important for improving accuracy as overlapping reads may have different alignment results in indel regions. Then base quality score recalibration needs to be performed to confirm the accuracy of Phred score and correct them if there are errors. After post-alignment processing, sequences are ready for variant calling.

2.3.3 Variant Calling

Variant calling is the identification of where aligned reads differ from a reference genome and writing to a VCF file, which is a widely used format for storing variant information. Due to different research purposes and various understanding of mutations, there is no standard category of variants. But in general, they can be classified into three categories by variation size: single nucleotide variant (SNV), indel and large structural variant (SV)[27, 28].

SNV is a replacement of one base by another base. It could be either a transition (interchange of the purine (Adenine/Guanine) or pyrimidine (Cytosine/Thymine) nucleic acids); or a transversion (interchange of a purine and pyrimidine nucleic acid). Short indels are very common in human genes, range from 1 to tens of bases. They are the second abundant form of human genetic variation[29]. SV usually includes copy number variant (CNV), translocation, inversion and other variants in the genome that may change the structure of coded proteins. CNV is the duplication of the same base or region in a genome[30]. Inversion is the flip of some region along the genome. Translocation is the exchange of sequences between two non-homologous chromosomes. Long indels are also considered as SVs.

According to the location of mutations, it can also be classified as germline mutation and somatic mutation. Germline mutations are inherited from parents. It arises in germ cells and can be incorporated into the DNA of every cell and passed to the offspring. In contrast, a somatic mutation is detected in a single cell and can not be inherited. It only affects tissues derived from the mutated cell.

Variant calling algorithms identify potential mutations by comparing mapped reads to the reference genome. But different programs may adopt different approaches. Techniques used in these algorithms will be covered in section 2.4. The results of variant calling are typically stored in a VCF file. A VCF file is composed of meta-information lines, a header line, and data lines. Meta-information lines describe the date of creation and file format and specify entries of columns in the data section. The header line names 8 required fields and may include other optional columns. Each line in data section represents a variant,

containing all information specified in the header line.

2.3.4 Annotation

Once variants are detected, we want to find their associations with some specific gene features. Variants need to be annotated with attributes such as genomic feature, gene symbol, exonic function, and amino acid change. Additional information, including minor allele frequency (MAF) in normal populations, experimental evidence from clinical assays, deleterious prediction of variant function, and collection of variants and genes in known cancer or genetic disease studies are also attached[31]. Most tools focus on the annotation of SNVs and indels in protein-coding region using information from publicly available database, like dbSNP, 1000 genome project, COSMIC, etc.

2.3.5 Visualization

In order to take full advantage of these genome sequence data and associated annotations, we sometimes need to transform test-based information into a picture to make sense of distribution features of data, which could help us understand the structure of our data.

2.3.6 Prioritization and Filtration

Variant calling usually produces thousands to millions variants. This large candidate list needs to be reduced to detect novel mutations from common polymorphisms. Filtration and prioritization include three steps[31, 32]. The first step is removal of less reliable variant calls, which includes variants with low quality, strand biased, located in SNV clusters, and/or supported by low-confidence read alignment. The second step is to restrict variants to those of relatively low population frequency, since a huge proportion of human diseases are caused by rare mutations[33]. The third step is to prioritize the variants related to the disease according to their predicted coding-effects. The last step is the most important one as the database-related annotation is limited to known disease and variants but most

variants could remain unknown. This step usually adopts machine learning algorithm or statistical methods to predict functional impact of variants. Only deleterious variants will be selected for further analysis.

2.3.7 Functional Effect Analysis

Further downstream analysis comes to the system biology of cancer cells, for example pathway analysis and network-based analysis. Studying interactions between components can improve the cancer treatment. This analysis can identify genes as a group that are directly associated with disease or pathological phenotypes[34].

2.4 Available Softwares Programs

Numerous bioinformatics and computational tools have been well established and applied in the variant analysis[20-63]. The underlying algorithms of these tools differ a lot and some comprehensive reviews have been carried out to compare and evaluate these tools. In this section, I will, for each step mentioned above, introduce several popular programs that can be applied. Also, some programs could incorporate several features and functionalities that are applied to different steps. At the end, some complete analytical pipelines will be discussed.

2.4.1 Quality Control Tools

Most sequencers will generate a quality report as part of their functionalities, but usually this can only identify problems caused by the sequencer itself. So we need some other tools to evaluate the quality of raw sequence data. FastQC[35] aims at providing a QC report that can point out problems originate either in the sequencer or in the starting library material. It accepts files from a variety of formats and assesses the quality of sequences by Phred score, GC content, PCR duplication rate, read depth and other statistical inferences.

It is compatible with all main sequencing platforms.

With the QC report, researchers can determine whether preprocessing steps need to be carried out. Cutadapt[36] is capable of finding and removing adapter sequences, primers, poly-A tails and other types of undesired sequence. Trimmomatic[37] includes a variety of processing steps for read trimming and filtering, but it is famous for handling paired-end reads.

2.4.2 Alignment Tool

Several alignment tools have been established including Bowtie and Bowtie2[38, 39], MAQ[40], BWA[41], SOAP, SOAP2 and SOAP3[42–44], Eland (Illumina suite), NextGenMap[45], rHAT[46]. Among these aligners, the three most commonly used are BWA, Bowtie (1 and 2) and SOAP(1, 2 and 3).

Bowtie(1 and 2), SOAP(2 and 3) and BWA all use Burrows-Wheeler Transformation techniques. Bowtie is a ultrafast, memory-efficient short read aligner that aligns 35-base pair (bp) reads at a rate of more than 25 million reads per CPU-hour. Bowtie2 extends the original version to allow gapped alignment using dynamic programming. It is also capable of aligning long reads. BWA is a short read aligner allowing mismatches and gaps. The updated version BWA-SW[47] can align long sequences up to 1 MB against a large sequence database with memory footprint less than 4 GB. SOAP is also a short read aligner that allows for gaps and mismatches. Three versions differ in alignment speed, memory usage and length of reads.

Picard(MarkDuplicates) and SAMtools[48] can be use to remove PCR duplicates. Picard is a set of command line tools written in Java. It identifies read duplicates according to their same 5' site coordinates and orientation. SAMtools is a library and software package for parsing and manipulating alignments in the SAM/BAM format[48]. It can not only remove duplicates, but also convert other alignment formats, sort alignments, call SNPs and small indels and view alignments in text.

2.4.3 Variant Calling Tools

Some popular variant calling programs for SNV and small indels include SAMtools, GATK [49] and VarScan2[50]. GATK, short for The Genome Analysis Toolkit, conducts germline variant calling based on bayesion approaches. It first produces the BAM file using BWA. After some necessary manipulations aforementioned, it calls variants for each sample read, then analyzes variants against known variants, and applies a calibration procedure to compute a false discovery rate for each variant. Variants are marked as homozygous (1/1) or heterozygous (0/1) in the sample column of the VCF file. The most updated version is GATK4, which incorporates Picard and extends functionality into copy number and somatic analyses and offers pipeline scripts for workflows. VarScan2 is a platform-independent mutation caller written in Java. It can take a tumor sample and the corresponding normal sample simultaneously and uses a heuristic and statistical algorithm to classify somatic status and detect CNVs in exome data from tumornormal pairs according to the read depth, base quality, variant allele frequency, and statistical significance.

FreeBayes[51] is a haplotype-based variant detector to find SNVs,indels and SVs for short sequences. To start, it needs an alignment file (BAM) and a reference genome (fasta). Then based on its Bayesian statistical framework, it detects and reports variants in a standard VCF file. A comparison of eight variant callers including the three mentioned here[52] suggests that FreeBayes has the best sensitivity when detecting variants with extreme allele frequencies.

2.4.4 Annotation Tools

Tools and packages for annotating variants include but are not limited to ANNOVAR [53], SnpEff[54], SnpSift[55], MuTect[56], VAT[57]. ANNOVAR is able to perform annotations of variants with different levels based on their functional effects, conserved genomic regions, predicted transcription factor binding sites, predicted microRNA target sites and predicted stable RNA secondary structures. To use the command-line driven software, one needs

to download required annotation datasets. The output shows the location of each variant with respect to genes and amino acid changes caused by the mutation. SnpEff annotates variants based on their genomic locations and predicts coding effects of genes. It contains a pre-built database with over 20000 reference genomes, but also allows users to build their own database if needed. SnpSift is always used together with SnpEff to filter out significant variants. MuTect applies a Bayesian classifier for detecting somatic mutations with very low allele fractions, requiring only a few supporting reads, followed by carefully tuned filters that ensure high specificity. VAT annotates variants from multiple personal genomes at the transcript level and obtain summary statistics across genes and individuals. The most distinguishable feature of VAT is its cloud computing capability, which provides tremendous storage, scalable compute resources and ability to share data for collaboration.

2.4.5 Visualization Tools

Visualization tools provide an interactive and intuitive graphical view of genomic data. A very popular tool is called Integrative Genomics Viewer (IGV)[58]. It supports flexible integration of many genomic data types including aligned sequences, mutations, copy number, gene expression, methylation, and genomic annotations. Another commonly used tool is UCSC genome browser.

2.4.6 Prioritization and Filtration Tools

Tools like VAAST2[59], VarSifer[60], KGGseq[61], have be developed to filter and prioritize variants. Aforementioned prediction tools are limited to known variants. Some prediction programs like PolyPhen-2[62], SIFT[63], FATHMM[64], PROVEAN[65], can be utilized for prediction effects of both known and novel variations.

2.4.6.1 PolyPhen2 and SIFT

PolyPhen-2, available as standalone software or via web server, predicts functional effects of non-synonymous SNVs in human using a naive Bayes classifier. The prediction is based on eight sequence and three structural features selected using an iterative greedy algorithm. PolyPhen-2 score ranges from 0 to 1, which suggests the probability that a substitution is deleterious. Variants with values closer to 0 are predicted to be benign and with values closer to 1 are damaging.

SIFT uses sequence homology-based approach to predict the impact of amino acid substitutions on protein function. SIFT is able to distinguish deleterious SNVs from neutral variants in human and does not require the information of protein structures. SIFT adopts a similar scoring scheme to PolyPhen-2, but with opposite meaning. It searches the databases for related protein sequences. The multiple alignment of query sequence from PSI-BLAST is converted into a position specific scoring matrix (PSSM). In this case, PSSM is an l $\times$ 20 matrix where l is the length of the protein sequence. Each entry, p_{ca}, is the probability of amino acid a at position c of the protein where c ranges from 1 to l and a is any one of the 20 amino acids. p_{ca} is estimated using the following formula[66]:

$$p_{ca} = \frac{N_c}{N_c + B_c} \cdot g_{ca} + \frac{B_c}{N_c + B_c} \cdot f_{ca} \tag{2.1}$$

where N_c is the total number of sequences in the alignment, g_{ca} is the sequence-weighted frequency that amino acid a appears at position c in the alignment, f_{ca} is the pseudo-count that amino acid a appears at position c and B_c is the total number of pseudo-count.

The score is then normalized on the amino acid with the highest p_{ca}. It represents the probability that the amino acid change is tolerated. A variant with a SIFT score close to zero (The cutoff is usually set at 0.05) is considered deleterious. Since the prediction only depends on the sequence, false positive rate could be increased if aligned sequences are closely related. So in the output they also provide a median sequence conservation value, ranges from 0 to 4.3, to show the diversity of aligned sequences. A warning will appear if the median conservation value is greater than 3.0, which means the confidence of the

prediction is low.

2.4.6.2 FATHMM

FATHMM is a sequence-based algorithm which combines evolutionary conservation in homologous sequences with pathogenicity weights, representing the overall tolerance of proteins. It can be applied to both human and nonhuman species. If an unweighted prediction is requested, the predicted magnitude of the effect on protein function is calculated using the following formula:

$$unweighted = \ln \frac{P_m/(1-P_m)}{P_w/(1-P_w)} \tag{2.2}$$

where P_w and P_m are underlying probabilities for the wild-type and mutant amino acid residues, respectively. For an improvement in human, a weighted prediction is used as follows:

$$weighted = \ln \frac{(1-P_w)(W_n+1)}{(1-P_m)(W_d+1)} \tag{2.3}$$

where W_d and W_n are the pathogenicity weights, representing the relative frequencies of disease-associated and functionally neutral amino acid substitutions mapping onto the relevant hidden Markov model, respectively.

Scores close to zero suggest that there is no significant effect in the underlying amino acid substitution. However, scores below zero suggest that the substitution is unfavorable and scores greater than zero suggest that a favorable substitution is observed.

2.4.6.3 PROVEAN

PROVEAN is an alignment-based algorithm that can predict the functional impact for all kinds of protein sequence variations (single amino acid substitutions, multiple substitutions and indels). It measures the change in sequence similarity of a query sequence to a protein sequence homolog before and after the introduction of an amino acid variation to the query sequence. This change in the alignment score is used as the implication of the effect of sequence variation.

In the first step, PROVEAN searches the databases for a set of clusters of homologous and distantly related sequences. Then a delta score, $\Delta(Q, v, S)$ is defined to represent the the change in sequence similarity. Here, Q is the query protein sequence and v is the variation with respect to its homologous sequence S. So,

$$\Delta(Q, v, S) = A(Q', S) - A(Q, S) \tag{2.4}$$

After calculating the delta score for each sequence, an average delta score is computed within each cluster. Again, the delta score is averaged among clusters. The equation is as follows:

$$score = \frac{1}{N} \sum_{c=1}^{N} (\frac{1}{N_c} \sum_{i=1}^{N_c} \Delta_{c,i}) \tag{2.5}$$

where N is the number of clusters in the supporting set, N_c is the number of supporting sequences in the c-th cluster, and $\Delta_{c,i}$ is the delta score of the i-th supporting sequence in the c-th cluster. This average is used as the final PROVEAN score. Low delta scores are interpreted as amino acid variations leading to a deleterious effect on protein function, while high delta scores are interpreted as variations with neutral effect on protein. The threshold is set as -2.5.

Programs introduced in this part are most popular ones. Extensive summaries of functional prediction tools can be found elsewhere[26, 67].

2.4.7 Resources for Functional Effect Analysis

Once candidate variants are identified, it is a common step to determine what biological processes these variants involve in and select genes that can be functionally linked to the pathways already known to be involved in the disease. Variants in genes that are totally unlinked to the known genes or pathways are largely neglected. Gene Ontology (GO) terms annotate genes according to their cellular components, molecular functions and biological processes. Pathway analysis provides concrete and detailed functional insight into the connection between genotype and phenotype[34, 68, 69]. Public databases like

KEGG[70], Reactome[71] and others are widely used resources. DAVID[72] is a popular web-based tool to provide functional interpretation of user defined gene list. Databases like DrugBank[73] can provide links between drugs and their targets (variants), which could help with exploring how functional variants affect drug efficacy and adverse effects. Protein-protein interaction network is a good way to explore relationships between different proteins.

2.4.8 Complete Analytical Pipelines

Each of the program mentioned above can perform one or several steps in the workflow of WGS analysis. But it could be inefficient if one wants to conduct the complete analysis. Fortunately, there are some pipelines that integrate these tools into a single workflow. Here I review three recent genome analysis pipelines Fastq2vcf[74], SpeedSeq[75] and ExScalibur[76].

Fastq2vcf performs the process of generating, annotating and analyzing sequence variants in a single or parallel computing environment, which distinguishes it from other pipelines. The users are allowed to specify program parameters and give a basic description of the sequence data. After configuration, three shell scripts will be generated which can automate analysis steps. FastQc and BWA are used for quality control and sequence alignment. After formatting the data (Convert SAM to BAM), post-processing , including duplicate marking, local realign and base quality score recalibration, are performed by employing Picard and GATK. The SAMtools, GATK and SNVer[77] are invoked to call variants and write results in a VCF file. Finally, annotation of variants will be carried out by ANNOVAR and VEP (ENSEMBLs Variant Effect Predictor[78].

SpeedSeq is an open-source platform that accomplishes alignment, variant detection and functional annotation in a very rapid speed. It uses BWA-MEN , which is one of the three algorithms in BWA package, to align paired-end FASTQ files to the human GRCh37 reference genome. The duplication marking is assigned to Samblaster[79]. Then FreeBayes is run to detect SNVs and indels. LUMPY, SVTyper (A maximum-likelihood Bayesian

classification algorithm developed by the same team) and CNVnator are used together to analyze and annotate SVs.

ExScalibur is a WES analysis pipeline that utilizes three alignment and six variant detection algorithms to perform accurate detections of germline and somatic mutations. The combination of aligners and callers can be specified by the user. It can be run on local computers, high-performance computing clusters and cloud environments. At the completion of each run, ExScalibur will generate a report which shows quality statistics, aggregates variants and estimates the concordance of all aligner/caller combinations.

Chapter 3

Materials and Methodology

In this chapter I will describe formally the methods we have used to develop the data preprocessing program and how we analyzed the performance of the program.

3.1 Materials

We downloaded 148 VCF files from The Cancer Genome Atlas. Each contains mostly protein coding sequence data from primary tumor tissue and normal tissue of a patient with PrCa. A 7 GB VCF file containing 29 samples (cancer patients, normal people and cell lines) were collected locally and used to test program's capability of handling large datasets. The file was reduced by different combinations of row and columns, forming 35 subsets of various sizes to analyze the runtime behavior.

The script was first tested on Biolinux07, a x64 Linux system with 4 cores and 16 GB memory using several PrCa datasets. Then, analyses of performance using all 148 small files and the large file and its subsets were conducted on our desktop server BinfCompute, a Linux machine Dell PowerEdge R730 with 32 cores and 256 GB memory. The program was written in Python3. Required Python modules and reference files are available on the machines.

3.2 Developing Preprocessing Tool

The OncoMiner preprocessing program is designed to handle the common VCF NGS files and convert them to the OMI format. The program is named *OP-VCF*.

3.2.1 Overall Workflow

The preprocessing tool is composed of three components which can be accomplished by various scripts. To improve efficiency, the function *splitvcf* and *splitgtf*, at the beginning, split the VCF file and the reference genome by chromosomes into smaller files for parallel processing. The overall workflow is depicted in Figure 3.1.

1. First, most of the required OMI fields can be extracted directly from the VCF file. This is done by a script called *writecsv*.

2. Second, the genomic region type needs to be determined by comparison with a reference genome containing precise locations for transcripts, exons, and coding regions. Script *vcf_func* can take the reference genome and VCF file as inputs and return a dictionary containing all variants and their genomic regions.

3. Then, the amino acid change types of variants within exome is determined by comparison with the original sequence and codons. This step is also accomplished by *vcf_func*.

4. The last function *writeOMI* combines these information and write a final OMI file.

The program can also record the number of chromosomes and variants, variants per chromosome and runtime for each VCF file.

3.2.2 GSV Identification

After splitting the VCF file and extracting most of required information (chromosome number, position, etc), the *parse_vcf* then proceeds to parse the GSV information stored in the VCF file. Variants are first filtered out if sequencing depth is less than five. Then, the classification of genomic region type for each variant is based on human genome assembly hg38 obtained from UCSC Genome Browser [80]. The reference information is stored in the form of a gene transfer format (GTF) file (Figure 3.2), which contains the start and

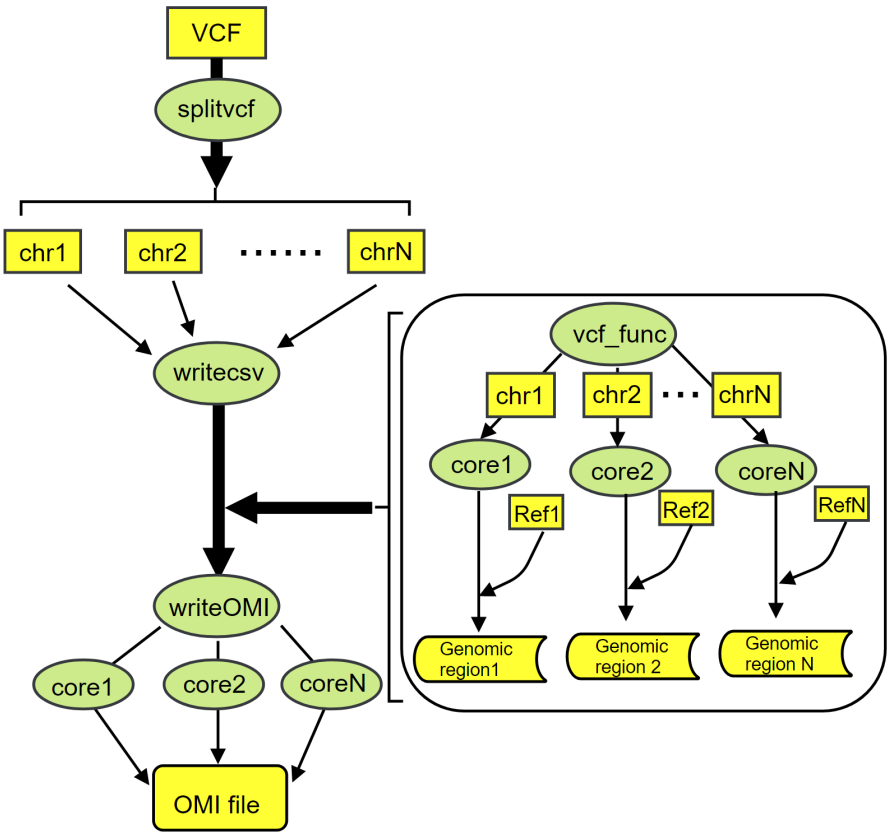

Figure 3.1: Workflow through which a VCF file is processed to an OMI file

end positions of all genes (4th and 5th column), coding regions (6th and 7th column) and exons (last 2 columns). According to this information, all GSVs can be classified using the decision tree shown in Figure 3.3[8]. Each unique GSV in the OMI file has an identifier consisting of the chromosome number and the position of the variant within the chromosome.

3.2.3 Determination of the Change Type of Amino Acid

For variants that are located within the exon, we need to determine whether the corresponding amino acid for which they code will be changed. We compared mutated sequence with the reference sequence. By looking up the dictionary for all 64 codons, the change type

```
DDX11L1 NR_046018       chr1    +       11873   14409   14409   14409   3       11873,12612,13220,      12227,12721,14409,
WASH7P  NR_024540       chr1    -       14361   29370   29370   29370   11      14361,14969,15795,16606,16857,17232,17605,17914,18267,24737,29320,
MIR6859-1       NR_106918       chr1    -       17368   17436   17436   17436   1       17368,  17436,
MIR6859-2       NR_107062       chr1    -       17368   17436   17436   17436   1       17368,  17436,
MIR6859-3       NR_107063       chr1    -       17368   17436   17436   17436   1       17368,  17436,
MIR6859-4       NR_128720       chr1    -       17368   17436   17436   17436   1       17368,  17436,
MIR1302-2       NR_036051       chr1    +       30365   30503   30503   30503   1       30365,  30503,
MIR1302-9       NR_036266       chr1    +       30365   30503   30503   30503   1       30365,  30503,
MIR1302-10      NR_036267       chr1    +       30365   30503   30503   30503   1       30365,  30503,
```

Figure 3.2: Example of a GTF file

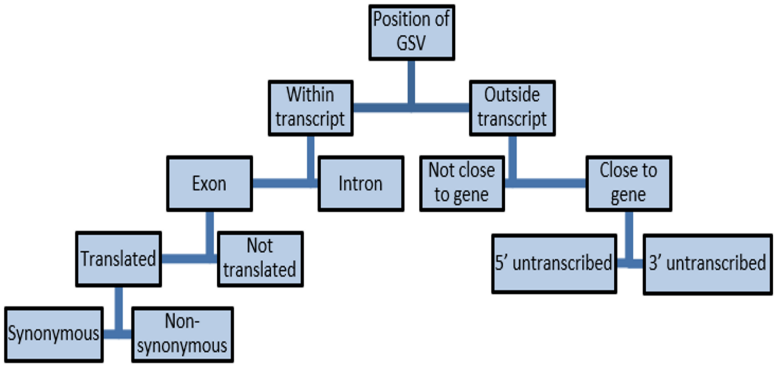

Figure 3.3: Decision tree. Variants are classified based on their positions in transcripts

of amino acids (synonymous or non-synonymous) could be identified. A function *writeOMI* then writes genomic region type and change type of amino acid (if applicable) to the OMI file to finalize the whole process.

3.2.4 Parallelization and Implementation

Our program makes use of the Python multiprocessing module to run the script in parallel. Each chromosome-specific VCF file will be assigned to a core. Users are allowed to specify the number of cores to be used. The only part that was in serial was *splitvcf* and *splitgtf*. Though the VCF file is processed by chromosome, each sample in the file will end with a complete OMI file containing all chromosomes in the orginal file. The *OP-VCF* was implemented on BinfCompute using various cores.

3.3 Analyzing Performance of the Script

Performance of parallelization was first evaluated by efficiency of core usage. We have recorded runtimes of 148 VCF files using various cores and calculated speedup(S_p) and efficiency(E_p) as S_p=T(1)/T(P) and E_p=S_p/p, where T(p) is the measured runtime using p cores with varying p = 1, 2, 4, 8, 16, and 24. Then, we split the original large VCF file into 35 subsets of various sizes consisting of different combinations of rows and columns. Rows were reduced rows to 1/10, 1/5, 1/4, 1/3, 1/2 of the original file respectively. Then for each generated file, columns (samples) were removed and left with 24, 19, 15, 10, 5, 1 respectively. These files were run with 8 cores on BinCompute to explore correlations of runtime with input file size, number of variants and number of samples. Statistical analysis of efficiencies and runtimes were performed using R.

Chapter 4

Results and Discussion

In this chapter I present the main result upon which this thesis is centered—runtimes of different files using various cores and behavior analysis of the program.

4.1 Feasibility of the Script

The *OP-VCF* program has been successfully implemented on both Biolinux07 and BinfCompute described in chapter 3. However, when we were testing large files on Biolinux07, some processes were either stopped by the system or entered an uninterruptable status. We monitored memory usage and noticed that all physical memory (16GB) was used together with a large fraction of swap memory. Therefore, the testing processing of large files was only conducted on BinfCompute since it has more memory. The program is able to take VCF files as input and produce OMI files that can be correctly accepted by OncoMiner.

Table 4.1: Average runtimes (in seconds) of different steps on BinfCompute using 1, 8 and 24 cores

Function	1 core	8 cores	24 cores
writecsv	2.39	0.37	0.13
vcf_func	10.20	2.02	1.19
writeOMI	4.83	0.69	0.28
Others	3.69	3.64	3.66
Total	21.04	6.70	5.26

We first ran some small files (from 441 KB to 910 KB) on both computers and found that runtime behaviors were similar. Then we moved on to test all 148 files on BinfCompute using 1,2,4,8,16 and 24 cores. The file size ranges from 244 KB to 1.95 MB. The overall runtime and runtimes of different steps are summarized in Table 4.1. For these small files, determination of genomic region type takes a large fraction (48.5% using 1 core) of the total time. However, this part is parallelized in the program and the decrease in runtime using more cores is substantial. The only part that needs to be run sequentially is the preparation for parallelization, *splitvcf* and *splitgtf* (runtime displayed as Others in table). However,parallelization of other steps could offset the time spent in splitting.

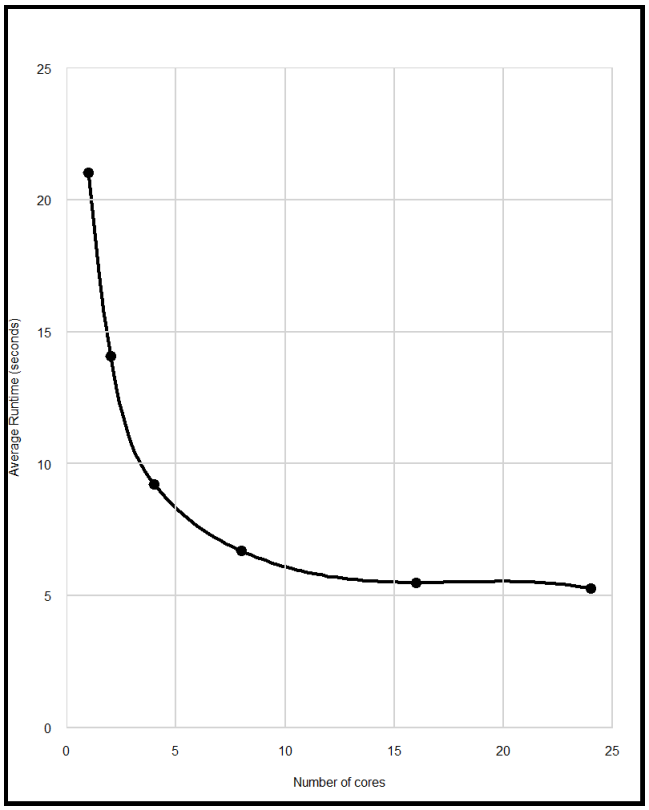

Figure 4.1: Average runtimes for test datasets on BinfCompute

Average runtimes using 1, 2, 4, 8, 16 and 24 cores are displayed in Figure 4.1. As the number of cores increases, the speedup is quite substantial. The highest speedup achieved was 4.001 using 24 cores. This is not surprising as Amdahl's law says that speedup of a

task at fix workload is always limited by the fraction of serial part F_s[81]. If we look at the runtime using 1 core, the serial part(shown as Others) is about 17.54% of the overall runtime. So, speedup using multi-cores can not exceed $1/F_s$, which is 5.7. Also, there were a lot of Input/Output in the program. Therefore, the actual speedup was below the desired one.

Also, we evaluated overall efficiencies of using multiple cores. To visualize efficiencies, we set up a baseline efficiency value for each core used. If no speedup was achieved, the baseline value would be 1/p, where p is the number of cores used. The dashed line in Figure 4.2 represents baseline efficiencies and solid line shows actual efficiencies. The best efficiency is achieved at 2 cores. This is not surprising as we know increasing number of cores always decreases efficiency.

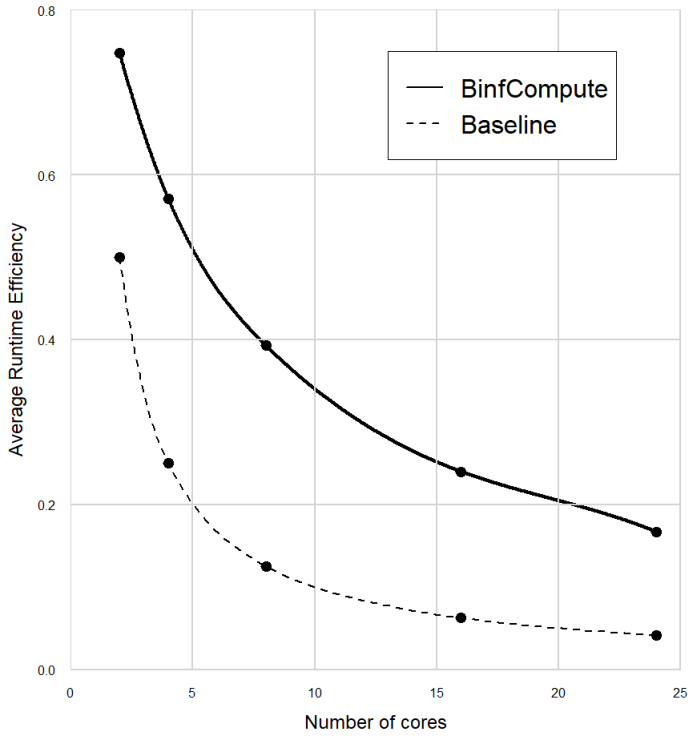

Figure 4.2: Average efficiencies for various cores on BinfCompute

The implementation of *OP-VCF* program on our local machine BinfCompute showed

good performance. Since VCF files were split and processed by chromosome, there were a lot of opening and closing of files (I/O). These waiting times resulted in a less desirable speedup set by Amdahl's law.

Next,we moved to large VCF files to analyze runtime behaviors. For this part we only used BinfCompute since the out of memory problem occurred on Biolinux07 for some of those files, which suggested that memory constraint could be a problem of our program. As runtimes of these files could be in the order of tens of hours, we decided to only tested them using 8 cores after comparing speedup and efficiencies of different cores.

4.2 Capability of Handling Large Datasets

Since the 7 GB VCF file is the only large file we obtained, we started to generate subfiles from the orginal one to explore how runtime performs with different sizes, rows or columns. We got 35 subfiles in total with file size ranging from 193.8 MB to 3.7 GB. Then we ran the program on these files using 8 cores. Runtimes varied from 35.7 minutes to 39.7 hours (see Figure 4.3).

From the graph, it is obvious that runtime follows a linear trend with respect to input file size. A simple linear regression suggested that the Pearson correlation coefficient r=0.9764 is significant (p value $< 2.2\times e^{-16}$). However, these subfiles were all generated from the same file, which contains around 3300 lines meta information. This part is the same in each file but was not processed by the program. This prompted us to explore more straightforward factors that may correlate with the runtime. We counted number of variants and samples in each subfile and looked into how they affect the runtime behavior. This time the correlation of runtime and number of samples was 0.570 with p value = $3.5\times e^{-4}$. Because the number of rows in each subfile was represented as the fraction of rows in the original file, we scaled them up to some small integers by multiplying each fraction by the common factor of denominators in all fractions. We also saw a significant positive correlation of runtime with number of variants (r=0.718, p value = $1.2\times e^{-6}$).

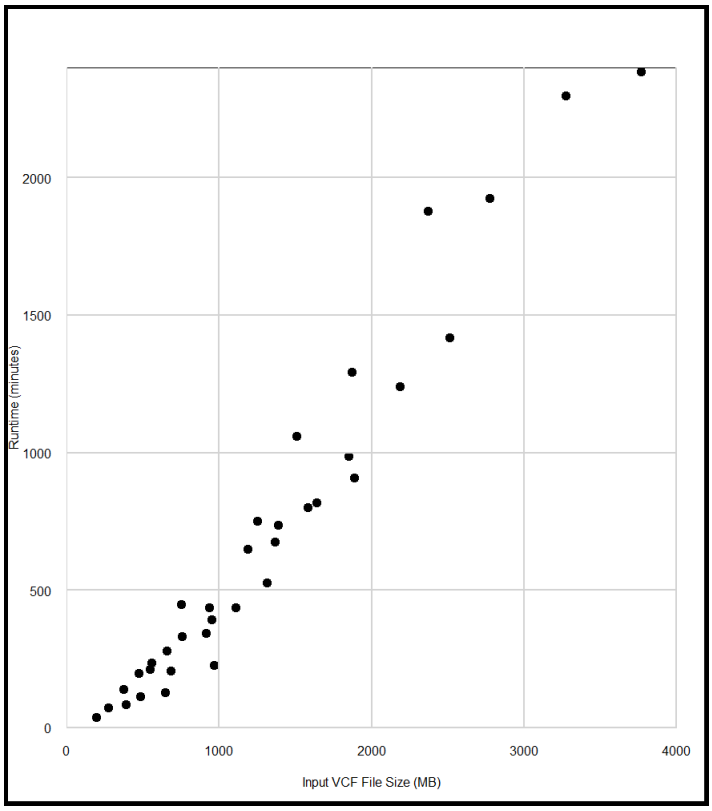

Figure 4.3: OP-VCF runtimes versus input file size

We noticed that on BinfCompute, the fraction of time spent on *writeOMI* was much longer compared with that for small files. Two possible reasons could be I/O issues and comparison. Since each sample is processed by chromosome, each core needs to access the OMI file and write outputs. This can cause some delay. Also, information extracted from VCF file needs to be compared with parsing information. Only if chromosome number and position in two parts match, they can be combined and written to output. As it requires a search in nested loop involving up to hundreds of thousands variants, the runtime in this part could be substantially long. In consideration of these factors, this runtime is not unacceptable but we still wish to cut it down further to make our program more efficient. To explore overall relationship of both factors with runtime, we fitted a multiple linear model to runtime with number of variants (scaled) and samples. The regression equation

is as follows:

$$Runtime = -787.2 + 0.44 \times (\#\ thousand\ variants) + 38.4 \times (\#\ samples) \tag{4.1}$$

The p value of F test was $1.7 \times e^{-13}$. It means the overall relationship is significant. Adjusted coefficient of determination R^2 was 0.83, indicating that 83% variation in the runtime can be explained by its relationship with number of variants and samples. This equation can help us estimate runtimes when we get new data files, which will result in a better planning of any future work that might involve large datasets.

Chapter 5

Conclusion and Future Work

In this chapter, I conclude my M.S. thesis project and propose problems for my Ph.D. dissertation research as future work.

5.1 Conclusion

Our preprocessing script *OP-VCF* is able to process NGS data in VCF format and produce OMI files that can be accepted by OncoMiner for downstream analysis. This script fills the gap between NGS data in VCF format and OncoMiner, and enables OncoMiner to analyze NGS data in the most common file formats. The multiprocessing approaches worked successfully in the functions *writecsv*, *vcf_func* and *writeOMI*. Parallelization effciencies were reasonable using multiple cores and can offset time spent in serial parts *splitvcf* and *splitgtf*.

5.2 Future Work

For future work, we will incorporate more functionalities into OncoMiner to make it a complete pipeline for NGS analysis. Figure 5.1 indicates how OncoMiner is fitted into the NGS analysis workflow. Then we will use OncoMiner on our PrCa and leukemia datasets to identify cancer-related variants and their corresponding genes, as predict their functional effects. By constructing functional interactions among genes, we aim at providing a scientific theoretical basis that can eventually help to design more precise and effective methods of diagnosis and treatment for these cancers.

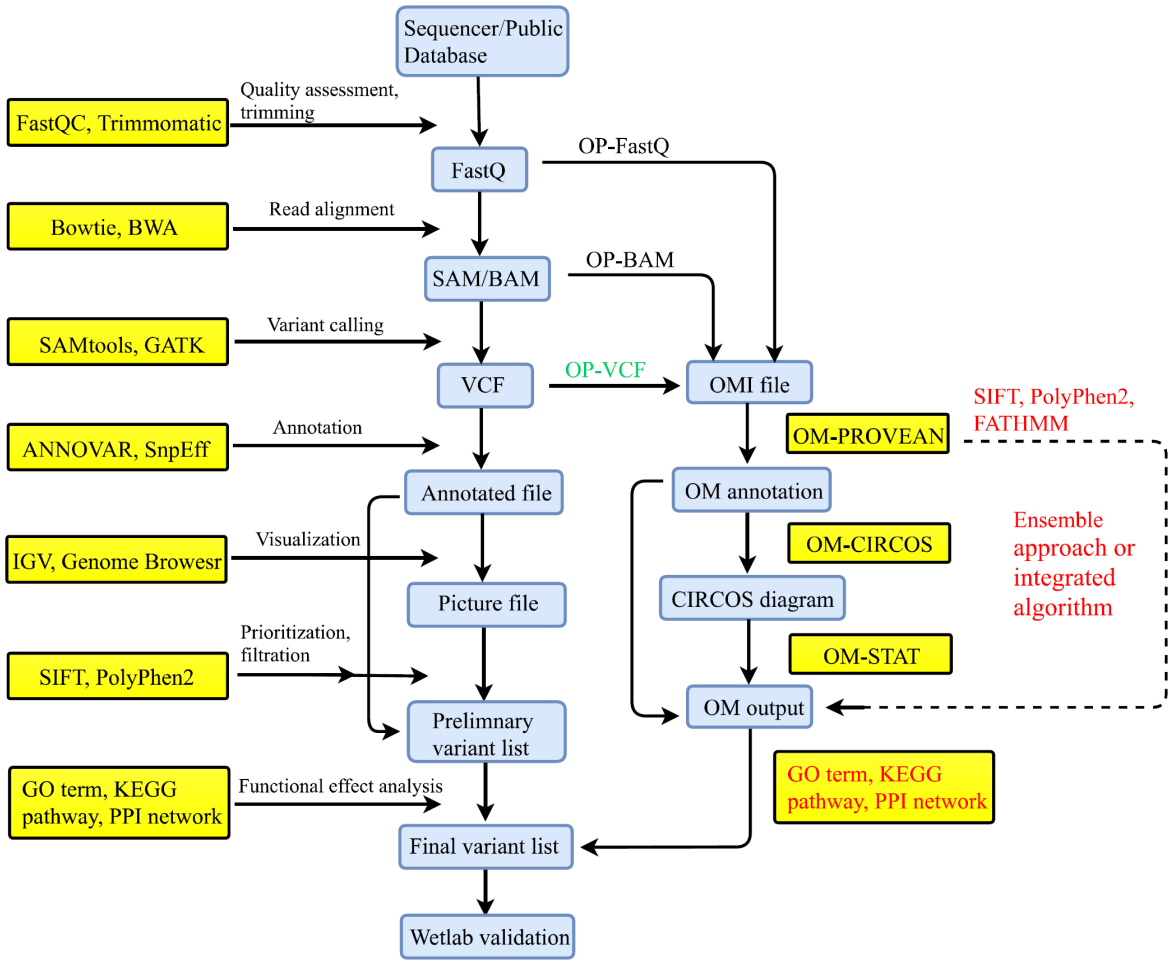

Figure 5.1: OncoMiner functionalities fitted into NGS workflow

5.2.1 Finalize the OncoMiner Preprocessing Program

As mentioned previously, we still plan to modify the *writOMI* function within the *OP-VCF* program to further cut down the runtime. Instead of comparing and writing variant by variant, I will try to first combine the information of all variants together in a list, and then write the whole list to the OMI file. In theory, this could increase the memory usage at some point. I plan to investigate into the balance of memory usage and runtime to optimize efficiency.

Now that we have demonstrated that our *OP-VCF* program works to produce correct

OMI files, and it is capable of processing large files in parallel. I will combine *OP-VCF* with the other two programs developed by our group for preprocessing FastQ and BAM/SAM files to complete the OncoMiner data preprocessing module. This module will be incorporated into the current OncoMiner pipeline, using the web framework web.py, on the UTEP High-Performance Cluster, HPC Version 3.2[7].

5.2.2 Analysis of Cancer Datasets

Once the OncoMiner preprocessing program is in place, we will be ready to analyze our entire collection of PrCa files from TCGA as well as the leukemia datasets. The compilation of all these datasets are now still in progress and is being done in collaboration with two undergraduate researchers in our group.

We download 503 VCF files from TCGA-PROD project, each representing one PrCa patient. Each case contains a primary tumor sample and either a normal blood sample or solid tissue sample. We will use OncoMiner to do comparative analysis of two samples in each case and filter out common variants which in general are presumed to be more likely to be neutral. Variants that have statistically significant difference between tumor and normal samples will be picked out for further analysis. One possible pitfall at this stage of my project is unexpected errors in various OncoMiner function might occur. This is not at all unusual when a recently developed piece of code is applied to a new set of data. I expect that some time will have to spent on debugging parts of the OncoMiner code.

Based on PROVEAN score calculated by OncoMiner, genes contain those potentially deleterious variants will be bioinformatically analyzed using DAVID[72] to perform the GO term annotation to annotate the unique biological functions. We can then use Kyoto Encyclopedia of Genes and Genomes (KEGG)[70] to analyze their signaling pathways. Enriched GO terms and KEGG pathways will be selected to analyze how mutant genes affect biological processes. To further investigate the potential links between these mutant genes, we are going to construct these genes into a protein-protein interaction (PPI) network using STRING[84]. Cytoscape[85] could be used to visualize interacted protein pairs. As

detected gene mutations may be potential biomarkers, we will report them for further experimental validation.

Also, we have a 7 GB VCF file, which was collected at local hospital and UTEP. This file contains 29 samples from ALL patients, cell lines, and normal individuals. Our group also have collected the data of 35 AML patients in BAM format. These datasets were downloaded from TCGA. The same strategy can be applied to this collection of datasets to detect leukemia related gene mutations, followed by downstream analyses as described in the previous paragraph.

5.2.3 Development of Ensemble approach/Integrated Algorithm

OncoMiner adopts PROVEAN score as a measure of the functional effects of protein sequence variations. It uses a "delta score" to represent alignment difference before and after the introduction of variant. There are currently other functional prediction tools like Polyphen-2, SIFT and FATHMM. They applied different machine learning methods and scoring schemes. A summary of these tools is displayed in Table 5.1. Their underlying algorithms have been reviewed in section 2.4.

Table 5.1: Basic information about 4 popular prediction tools

Prediction tool	Score range	Cutoff of deleteriousness	Method
Polyphen-2	0 - 1	>0.5	Naive Bayes classifier
SIFT	0 - 1	<0.5	Homology-based approach
FATHMM	0 - 1	>0.5	Hidden Markov model
PROVEAN	-40 - 20	<some threshold(eg.-2.5)	Alignment-based delta score

Many studies[83, 86] have compared the effectiveness of these widely used pathogenic variant predictors. Although each tool may outperform others in some aspect, some studies[82, 83] indicate that the accuracies of contemporary functional prediction tools are likely to be considerably lower than reported in their original method publications. I will

compare the results of OncoMiner, which uses the PROVEAN alignment-based algorithm, against those from other common prediction tools. We will consult experienced biomedical researchers who can help us by reviewing the resulting GSVs to provide expert opinions regarding the prediction accuracies of the different algorithms. From these I will attempt to create either an ensemble approach or an integrated algorithm that can achieve higher prediction accuracy. After incorporating the new algorithm into OncoMiner pipeline, it can be further tested on other cancer datasets as well.

5.2.4 Timeline

Table 5.2 summaries the tentative timeline for future work to be completed.

Table 5.2: Timeline for the completion of this work

Date	Tasks to complete	Goals
May. - Aug. 2019	Incorporate the script into OncoMiner	Resolve potential errors
Sept. - Dec. 2019	Analyze GSVs in PrCa and AML	Poster presentation at a conference
Jan. - May. 2020	Compare different prediction tools	Prepare first paper
Jun. - Aug. 2020	Develop an integrated algorithm	Submit first paper
Sept. - Dec. 2020	Incorporate the algorithm into OncoMiner	Give a talk at a conference
Jan. - May. 2021	Analyze variants in non-coding regions	Dissertation and second manuscript
Jun. - Aug. 2021	Finalize dissertation	Dissertation defense and paper published

References

[1] P. A. Futreal, L. Coin, M. Marshall, et al, "OncoMiner: A Pipeline for Bioinformatics Analysis of Exonic Sequence Variants in Cancer," *A census of human cancer genes*, 2004, Vol. 4, NO.3, pp. 177–183.

[2] American Cancer Society. Cancer Statistics Center. http://cancerstatisticscenter.cancer.org. Accessed Mar. 23, 2019.

[3] R.L . Grubb, "Prostate Cancer: Update on Early Detection and New Biomarkers," *Missouri Medicine*, 2018, Vol. 115, Issue. 2, pp. 132–134.

[4] W. B. Isaacs and J. Xu, "Current progress and questions in germline genetics of prostate cancer" *Asian Journal of Urology*, 2019, Vol. 6, Issue 1, pp. 3–9.

[5] T. H. Tran M. H. Harris, J. V. Nguyen, et al, "Prognostic impact of kinase-activating fusions and IKZF1 deletions in pediatric high-risk B-lineage acute lymphoblastic leukemia" *Blood Advances*, 2018, Vol. 2, Issue 5, pp. 529–533.

[6] T. Brito, E. Roger, S. Thpot, et al, "Advances in treatment formulations for acute myeloid leukemia," *Drug Discovery Today*, 2018, Vol. 23, No. 12, pp. 1936–1949.

[7] M. Y. Leung, J. A. Knapka, A. E. Wagler, et al, "OncoMiner: A Pipeline for Bioinformatics Analysis of Exonic Sequence Variants in Cancer," *Big Data Analytics in Genomics*, 2016, Springer, New York, NY, pp. 373–396.

[8] M. Vasquez, J. Mohl, MY. Leung, "Parsing Next Generation Sequencing Data in Parallel Environments for Downstream Genetic Variation Analysis," *The Journal of Computational Science Education*, 2018, Vol. 9, Issue. 2, pp. 37–45.

[9] R. L. Siegel, K. D. Miller, A. Jemal, et al, "Cancer statistics, 2018" *CA: A Cancer Journal for Clinicians*, 2018, Vol. 68, Issue. 1, pp. 7–30.

[10] C. H. Pernar, E. M. Ebot, K. M. Wilson, et al, "The Epidemiology of Prostate Cancer" *Cold Spring Harbor Perspectives in Medicine*, 2018, Vol. 8, Issue. 12.

[11] S. L. Zheng, J. Sun, F. Wiklund, et al, "Cumulative association of five genetic variants with prostate cancer" *The New England Journal of Medicine*, 2008, Vol. 358, No. 9, pp. 910–919.

[12] B. T. Helfand, A. J. Fought, S. Loeb, et al, "Genetic prostate cancer risk assessment: common variants in 9 genomic regions are associated with cumulative risk." *The Journal of Urology*, 2010, Vol. 184, No. 2, pp. 501–505.

[13] N. Short, M. Rytting and J. Cortes, "Acute Myeloid Leukaemia," *The Lancet*, 2018, Vol. 392, Issue. 10147, pp. 593–606.

[14] The Cancer Genome Atlas Research Network, "Genomic and epigenomic landscapes of adult de novo acute myeloid leukemia," *The New England Journal of Medicine*, 2013, Vol. 368, No. 22, pp. 2059–2074.

[15] E. Papaemmanuil, P. Paschka, N. Bolli, et al, "Genomic Classification and Prognosis in Acute Myeloid Leukemia" *The New England Journal of Medicine*, 2016, Vol. 374, No. 23, pp. 2209–2221.

[16] D. A. Siegel, S. J. Henley, J. Li, et al, "Rates and Trends of Pediatric Acute Lymphoblastic Leukemia United States, 20012014," *Morbidity and Mortality Weekly Report*, 2017, Vol. 66, No. 36, pp. 950–954.

[17] R. Okamoto, S. Ogawa, D. Nowak, et al, "Genomic profiling of adult acute lymphoblastic leukemia by single nucleotide polymorphism oligonucleotide microarray and comparison to pediatric acute lymphoblastic leukemia," *Haematologica*, 2010, Vol. 95, Issue. 9, pp. 1481–1488.

[18] H. Xu, W. Yang, V. A. Perez, et al, "Novel susceptibility variants at 10p12.31-12.2 for

childhood acute lymphoblastic leukemia in ethnically diverse populations," *Journal of the National Cancer Institute*, 2013, Vol. 105, Issue. 10, pp. 733–742.

[19] A. Gutierrez, M. Guerrero, V. Dolzan, et al, "Involvement of SNPs in miR-3117 and miR-3689d2 in childhood acute lymphoblastic leukemia risk," *Oncotarget*, 2018, Vol. 9, Issue. 33, pp.22907–22914.

[20] F. Sanger and A. R. Coulson, "A rapid method for determining sequences in DNA by primed synthesis with DNA polymerase" *Journal of Molecular Biology*, 1975, Vol. 94, Issue. 3, pp. 441–446.

[21] J. Reuter, D. Spacek and M. Snyder, "High-Throughput Sequencing Technologies" *Molecular Cell*, 2015, Vol. 58, Issue. 4, pp. 586–597.

[22] M. Choi, U. Scholl, W. Ji, et al "Genetic diagnosis by whole exome capture and massively parallel DNA sequencing" *Proceedings of the National Academy of Science*, 2009, Vol. 106, No. 45, pp. 19096–19101.

[23] N. Meena, P. Mathur, K. M. Medicherla,et al, "A Bioinformatics Pipeline for Whole Exome Sequencing: Overview of the Processing and Steps from Raw Data to Downstream Analysis" *Systems Biology*, 2018, Bio101: e2805.

[24] B. Ewing and P. Green, "Base-Calling of Automated Sequencer Traces Using Phred. II. Error Probabilities" *Genome Research*, 1998, Vol. 8, pp. 186–194.

[25] S. Pabinger, A. Dander, M. Fischer, et al, "A survey of tools for variant analysis of next-generation genome sequencing data" *Briefings in Bioinformatics*, 2013, Vol. 15, Issue. 2, pp. 256–278.

[26] J. D. Hintzsche, W. A.Robinson and A. C. Tan, "A Survey of Computational Tools to Analyze and Interpret Whole Exome Sequencing Data" *International Journal of Genomics*, 2016, Vol. 2016, ID. 7983236.

[27] C. Xu, "A review of somatic single nucleotide variant calling algorithms for next-generation sequencing data," *Computational and Structural Biotechnology Journal*, 2018, Vol. 16, pp. 15–24.

[28] A. Sathyanarayanan, S. Manda, M. Poojary, et al, *Encyclopedia of Bioinformatics and Computational Biology*, Elsevier, 2018

[29] J. M. Mullaney, R. E. Mills, W. S. Pittard, et al, "Small insertions and deletions (INDELs) in human genomes," *Human Molecular Genetics*, 2010, Vol. 19, Issue. R2, pp. R131–R136.

[30] G. Escarams, E. Docampo, and R. Rabionet, "A decade of structural variants: description, history and methods to detect structural variation," *Briefings in Functional Genomics*, 2015, Vol. 14, Issue. 5, pp. 305–314.

[31] R. Bao, L. Huang, J. Andrade, et al, "Review of Current Methods, Applications, and Data Management for the Bioinformatics Analysis of Whole Exome Sequencing," *Cancer Informatics*, 2014, Vol. 13, pp. 67–82.

[32] M. A. Ergun, A. Unal, S. G. Ergun, et al, "A new method for analysis of whole exome sequencing data (SELIM) depending on variant prioritization," *Informatics in Medicine Unlocked*, 2017, Vol. 8, pp. 51–53.

[33] J. McClellan, MC. King, "Genetic Heterogeneity in Human Disease," *Cell*, 2010, Vol. 141, Issue. 2, pp. 210–217.

[34] J. Li, A. M. N. Batcha, B. Grning, et al, "An NGS Workflow Blueprint for DNA Sequencing Data and Its Application in Individualized Molecular Oncology," *Cancer Informatics*, 2015, Vol. 14, Issue. s5, pp. 87–107.

[35] S. Andrews, *FastQC*, Babraham Bioinformatics, Cambridge, UK, 2012.

[36] M. Martin, "Cutadapt removes adapter sequences from high-throughput sequencing reads," *EMBnet* , 2011, Vol. 17, NO. 1, pp. 10–12.

[37] A. M. Bolger, M. Lohse, and B. Usadel, "Trimmomatic: a flexible trimmer for Illumina sequence data.," *Bioinformatics*, 2014, Vol. 30, Issue. 15, pp. 2114–2120.

[38] B. Langmead, C. Trapnell, M. Pop, et al, "Ultrafast and memory-efficient alignment of short DNA sequences to the human genome," *Genome Biology*, 2009, Vol. 10, NO. 3, pp. R25.

[39] B. Langmead, and S. L. Salzberg, "Fast gapped-read alignment with Bowtie 2," *Nature Methods*, 2012, Vol. 9, NO. 4, pp. 357–359.

[40] H. Li, J. Ruan and R. Durbin, "Mapping short DNA sequencing reads and calling variants using mapping quality scores," *Genome Research*, 2008, Vol. 18, NO. 11, pp. 1851–1858.

[41] H. Li, and R. Durbin, "Fast and accurate short read alignment with BurrowsWheeler transform," *Bioinformatics*, 2009, Vol. 25, Issue. 14, pp. 1754–1760.

[42] R. Li, Y. Li, K. Kristiansen, et al, "SOAP: short oligonucleotide alignment program," *Bioinformatics*, 2008, Vol. 25, Issue. 5, pp. 713–714.

[43] R. Li, C. Yu, Y. Li, et al, "SOAP2: an improved ultrafast tool for short read alignment," *Bioinformatics*, 2009, Vol. 25, Issue. 15, pp. 1966–1967.

[44] C. Liu, T. Wong, E. Wo, et al, "SOAP3: ultra-fast GPU-based parallel alignment tool for short reads," *Bioinformatics*, 2012, Vol. 28, Issue. 6, pp. 878–879.

[45] F. Sedlazeck, P. Rescheneder, A. Haeseler, "NextGenMap: fast and accurate read mapping in highly polymorphic genomes," *Bioinformatics*, 2013, Vol. 29, Issue. 21, pp. 2790–2791.

[46] B. Liu, D. Guan, M. Teng, et al, "rHAT: fast alignment of noisy long reads with regional hashing," *Bioinformatics*, 2016, Vol. 32, Issue. 11, pp. 1625–1631.

[47] H. Li, and R. Durbin, "Fast and accurate long-read alignment with BurrowsWheeler transform," *Bioinformatics*, 2010, Vol. 26,Issue. 5, pp. 589–595.

[48] H. Li, B. Handsaker, A. Wysoker, et al, "The Sequence Alignment/Map format and SAMtools," *Bioinformatics*, 2009, Vol. 25, Issue. 16, pp. 2078–2079.

[49] A. McKenna, M. Hanna, E. Banks, et al, "The Genome Analysis Toolkit: A MapReduce framework for analyzing next-generation DNA sequencing data," *Genome Research*, 2010, Vol. 20, NO. 9, pp. 1297–1303.

[50] D. C. Koboldt, Q. Zhang, D. E. Larson, et al, "VarScan 2: Somatic mutation and copy number alteration discovery in cancer by exome sequencing," *Genome Research*, 2012, Vol. 22, NO. 3, pp. 568–576.

[51] E. Garrison, and G. Marth, "Haplotype-based variant detection from short-read sequencing," *ArXiv12073907 Q-Bio*, 2012.

[52] E. Sandmann, A. O. Graaf, M. Karimi, et al, "Evaluating Variant Calling Tools for Non-Matched Next-Generation Sequencing Data," *Scientific Reports*, 2017, Vol. 7, Article NO. 43169.

[53] K. Wang, M. Li, H. Hakonarson, "ANNOVAR: Functional annotation of genetic variants from next-generation sequencing data," *Nucleic Acids Research*, 2010, Vol. 38, Issue. 16, pp. e164.

[54] P. Cingolani, A. Platts, L. Wang, et al, "A program for annotating and predicting the effects of single nucleotide polymorphisms, SnpEff: SNPs in the genome of Drosophila melanogaster strain," *Fly*, 2012, Vol. 6, Issue. 2, pp. 80–92.

[55] P. Cingolani, V. M. Patel, M. Coon, et al, "Using Drosophila melanogaster as a model for genotoxic chemical mutational studies with a new program, SnpSift," *Frontiers in Genetics*, 2012, Vol. 3, Article. 35.

[56] K. Cibulskis, M. S. Lawrence, s. L. Carter, et al, "Sensitive detection of somatic point mutations in impure and heterogeneous cancer samples," *Nature Biotechnology*, 2013, Vol. 31, Issue. 3, pp. 213–219.

[57] L. Habegger, S. Balasubramanian, D. Chen, et al, "VAT: a computational framework to functionally annotate variants in personal genomes within a cloud-computing environment," *Bioinformatics*, 2012, Vol. 28, Issue. 17, pp. 2267–2269.

[58] J. T. Robinson, H. Thorvaldsdttir, W. Winckler, et al, "Integrative Genomics Viewer," *Nature Biotechnology*, 2011, Vol. 29, Issue. 1, pp. 24–26.

[59] H. Hu, C. D. Huff, B. Moore, et al, "VAAST 2.0: Improved Variant Classification and DiseaseGene Identification Using a ConservationControlled Amino Acid Substitution Matrix," *Genetic Epidemiology*, 2013, Vol. 37, NO. 6, pp. 622–634.

[60] J. K. Teer, E. D. Green, J. C. Mullikin, et al, "VarSifter: Visualizing and analyzing exome-scale sequence variation data on a desktop computer," *Bioinformatics*, 2012, Vol. 28, Issue. 4, pp. 599–600.

[61] M. Li, H. Gui, J. S. Kwan, et al, "A comprehensive framework for prioritizing variants in exome sequencing studies of Mendelian diseases," *Nucleic Acids Research*, 2012, Vol. 40, Issue. 7, pp. e53.

[62] I. A. Adzhubei, S. Schmidt, L. Peshkin, et al, "A method and server for predicting damaging missense mutations," *Nature Methods*, 2010, Vol. 7, Issue. 4, pp. 248–249.

[63] P. C. Ng, and S. Henikoff, "SIFT: predicting amino acid changes that affect protein function," *Nucleic Acids Research*, 2003, Vol. 31, Issue. 13, pp. 3812–3814.

[64] H. A. Shihah, J. Gough, D. N. Cooper, et al, "Predicting the Functional, Molecular, and Phenotypic Consequences of Amino Acid Substitutions using Hidden Markov Models," *Human Mutation*, 2012, Vol. 34, Issue. 1, pp. 57–65.

[65] Y. Choi, G. E. Sims, S. Murphy, et al, "Predicting the Functional Effect of Amino Acid Substitutions and Indels," *PLoS ONE*, 2012, Vol. 7, Issue. 10, pp. e46688.

[66] P. C. Ng, and S. Henikoff, "Predicting Deleterious Amino Acid Substitutions," *Genome Research*, 2001, Vol. 28, Issue. 11, pp. 863–874.

[67] G. R. Ritchie, and P. Flicek, "Computational approaches to interpreting genomic sequence variation," *Genome Medicine*, 2014, Vol. 6, Issue. 10, pp. 87.

[68] I. Kuperstein, L. Grieco, D. P. Cohen, et al, "The shortest path is not the one you know: application of biological network resources in precision oncology research," *Mutagenesis*, 2015, Vol. 30, Issue. 2, pp. 191–204.

[69] P. Creixell, J. Grieco, S. Haider, et al, "Pathway and network analysis of cancer genomes," *Nature Methods*, 2015, Vol. 12, Issue. 7, pp. 615–621.

[70] M. Kanehisa, and S. Goto, "KEGG: Kyoto Encyclopedia of Genes and Genomes," *Nucleic Acids Research*, 2000, Vol. 28, Issue. 1, pp. 27–30.

[71] D. Croft, G. O'kelly, G. Wu, et al, "Reactome: a database of reactions, pathways and biological processes," *Nucleic Acids Research*, 2011, Vol. 39, Issue. 1, pp. 691–697.

[72] D. Huang, B. T. Sherman, Q. Tan, et al, "DAVID Bioinformatics Resources: expanded annotation database and novel algorithms to better extract biology from large gene lists," *Nucleic Acids Research*, 2007, Vol. 35, Issue. suppl_2, pp. W169–W175.

[73] V. Law, C. Knox, Y. Djoumbou, et al, "DrugBank 4.0: shedding new light on drug metabolism," *Nucleic Acids Research*, 2014, Vol. 42, Issue. D1, pp. D1091–D1097.

[74] X. Gao, J. Xu and J. Starmer, "Fastq2vcf: a concise and transparent pipeline for whole-exome sequencing data analyses," *BMC Research Notes*, 2015, Vol. 8, Issue. 72, pp. 3812–3814.

[75] C. Chiang, R. M. Layer, G. G. Faust, et al, "SpeedSeq: ultra-fast personal genome analysis and interpretation," *Nature Methods*, 2015, Vol. 12, N0. 1, pp. 72.

[76] R. Bao, K. Hernandez, L. Huang, et al, "ExScalibur: A High-Performance Cloud-Enabled Suite for Whole Exome Germline and Somatic Mutation Identification," *PLoS ONE*, 2015, Vol. 10, Issue. 8, pp. e0135800.

[77] Z. Wei, W. Wang, P. Hu, et al, "SNVer: a statistical tool for variant calling in analysis of pooled or individual next-generation sequencing data," *Nucleic Acids Research*, 2011, Vol. 39, Issue. 19, pp. e132.

[78] W. McLaren, B. Pritchard, D. Rios, et al, "Deriving the consequences of genomic variants with the Ensembl API and SNP Effect Predictor," *Bioinformatics*, 2010, Vol. 26, Issue. 16, pp. 2069–2070.

[79] G. G. Faust and I. M. Hall, "SAMBLASTER: fast duplicate marking and structural variant read extraction," *Bioinformatics*, 2014, Vol. 30, Issue. 17, pp. 2503–2505.

[80] J. Casper, A. S. Zweig, C. Villarreal, et al, "The UCSC Genome Browser database: 2018 update," *Nucleic Acids Research*, 2018, Vol. 46, Issue. D1, pp. D762–D769.

[81] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," *Proceedings of the AFIPS '67 Spring Joint Computer Conference*, Atlantic City, New Jersey, USA. April 18–20, 1967. Washington D.C.:Thomson Book Company.

[82] K. Mahmood, C. Jung, G. Philip, et al, "Variant effect prediction tools assessed using independent, functional assay-based datasets: implications for discovery and diagnostics," *Human Genomics*, 2017, Vol. 11, Issue. 10.

[83] D. G. Vasquez, C. A. Azencott, F. Aicheler, et al, "The Evaluation of Tools Used to Predict the Impact of Missense Variants Is Hindered by Two Types of Circularity," *Human Mutation*, 2015, Vol. 36, Issue. 5, pp. 513–523.

[84] D. zklarczyk, A. Franceschini, S. Wyder, et al, "STRING v10: proteinprotein interaction networks, integrated over the tree of life," *Nucleic Acids Research*, 2015, Vol. 43, Issue. D1, pp. D447–D452.

[85] P. Shannon, A. Markiel, O. Ozier, et al "Cytoscape: a software environment for integrated models of biomolecular interaction networks," *Genome Research*, 2003, Vol. 13, Issue. 11, pp. 2498–2504.

[86] M. S. Hassan, A. A. Shaalan, M. I. Dessouky, et al "Evaluation of computational techniques for predicting non-synonymous single nucleotide variants pathogenicity," *Genomics*, 2018, in press.

Appendix A

OP-VCF Program

A.1 Main Function and Generating Subfiles

```
from parse_VCF import vcf_func
from VCF2CSV import writecsv
from functools import partial
import csv
import vcf
import gc
import time
import sys
import os
import pdb
import xlwt
from xlutils.copy import copy
from xlrd import open_workbook
import multiprocessing
from functools import partial
import resource

#Function combines all information and writes an OMI file
def writeOMI(infile,outfile,newfile):
    start=time.time()
```

```
    f1=open(outfile,'r')
    rows=csv.reader(f1)
    f2=open(newfile,'a',newline='')
    writer=csv.writer(f2)
    for row in rows:
        for key in infile:
            if (row[1]==infile[key][0] and row[2]==str(key)):
                for i in range(len(infile[key][1])):
                    row.insert(2,infile[key][1][i])
                    row.append(infile[key][2][i])
                    row.append(infile[key][3][i])
                    writer.writerow(row)
                    del row[2]
                    del row[-2]
                    del row[-1]
                break

    f1.close()
    f2.close()
    end=time.time()-start
    del rows
    gc.collect()

#Function that splits VCF file by chromosome for parallel processing
def splitvcf(vcffile):
    start=time.time()
    for i in range(1,23):
        vcf_reader = vcf.Reader(open('%s' %(vcffile), 'r'))
```

```
        outvcf=open('chr'+str(i)+'.vcf','w')#names['chr%s' %i]=[]
        my_out=vcf.Writer(outvcf,vcf_reader)
        for record in vcf_reader:
            if (record.CHROM== 'chr'+str(i)):
                my_out.write_record(record)
        outvcf.close()
    vcf_reader = vcf.Reader(open('%s' %(vcffile), 'r'))
    outvcf1=open('chrX.vcf','w')
    outvcf2=open('chrY.vcf','w')
    my_out1=vcf.Writer(outvcf1,vcf_reader)
    my_out2=vcf.Writer(outvcf2,vcf_reader)

    for record in vcf_reader:
        if (record.CHROM== 'chrX'):
            my_out1.write_record(record)
        elif(record.CHROM== 'chrY'):
            my_out2.write_record(record)
        else:
            continue
    outvcf1.close()
    outvcf2.close()
    end=time.time()-start
    del vcf_reader
    gc.collect()
    print('Time for generating vcf files:'+str(round(end))+'s')
    return end

#Function that splits reference file by chromosome for parallel processing
```

```
def splitgtf(gtfdir):
    start=time.time()
    filegtf=open(gtfdir,'r')
    flat=filegtf.readlines()
    filegtf.close()
    for i in range(1,23):
        myout=open('chr'+str(i)+'_ref.txt','a')
        myout1=open('chrX_ref.txt','a')
        myout2=open('chrY_ref.txt','a')
        for line in flat:
            if (line.split()[2]== 'chr'+str(i)):
                myout.write(line)
            elif(line.split()[2]== 'chrX'):
                myout1.write(line)
            elif(line.split()[2]== 'chrY'):
                myout2.write(line)
            else:
                continue
        myout.close()
        myout1.close()
        myout2.close()
    end=time.time()
    del flat
    gc.collect()
    print('Time for generating gtf files:'+str(round(end-start))+'s')

#Function that counts total number of variants, chromosomes and
#variants per chromosome
```

```
def count_chrom(vcfdir):
    vcffile=open('%s'%(vcfdir), 'r')
    vcfall = vcffile.readlines()
    chrom=[]
    var_chr=[]
    var_total=0
    count=0
    chrpool=['chrX','chrY']
    for i in range(1,23):
        chrpool.insert(i-1,'chr'+str(i))
    for k in range(len(vcfall)):
        line = vcfall[k].strip()
        vl = line.split()
        if(vl[0] in chrpool):
            var_total+=1
            if(vl[0] not in chrom):
                chrom.append(vl[0])
                var_chr.append(count)
                count=1
            else:
                count+=1
    var_chr.append(count)

    del var_chr[0]
    info=[chrom,var_chr,var_total]
    vcffile.close()
    del vcfall
    del chrpool
```

```
    gc.collect()
    return info

if __name__=='__main__':
    vcfdir = sys.argv[1] #directory of input file
    gtfdir = sys.argv[2] #directory of ref file
    cores = int(sys.argv[3]) #number of cores to be used
    vcfs=os.listdir(vcfdir)
    #remove non-vcf files under vcfdir
    vcfs=[value for value in vcfs if value.endswith('.vcf')]
    (refpath, gtfname) = os.path.split(gtfdir)
########## create the excel file to record runtimes if file does not exist
    if(not os.path.exists('runtime.xls')):
        workbook=xlwt.Workbook()
        sheet1=workbook.add_sheet('sheet1',cell_overwrite_ok=True)
        colname=['filename','Size','# GSVs','# chroms','# of samples']
        for i in range(len(colname)):
            sheet1.write(0,i,colname[i])
        workbook.save('runtime.xls')
########## deal with each file
    for file in vcfs:
        #memory usage at the beginning
        st_mem=resource.getrusage(resource.RUSAGE_SELF).ru_maxrss/1024
        starting = time.time()
        time0=splitvcf(vcfdir+'/'+file)
        splitgtf(gtfdir)

        info=count_chrom(vcfdir+'/'+file)
```

```
    (filename,extension)=os.path.splitext(file)
    filesize=os.path.getsize(vcfdir+'/'+file)/(1024*1024)
    if ('chrM' in info[0]):
        info[0].remove('chrM')
    vcf_by_chrom=[b+'.vcf' for b in info[0]]
    time1=time.time()
######## create multiprocessing pool for copying information
    pool=multiprocessing.Pool(processes=cores,maxtasksperchild=1)
    pool_list1=[]
    pool_list1=[pool.apply_async(writecsv,(vcf_by_chrom[i],))
                for i in range(len(info[0]))]
    result1=[c.get() for c in pool_list1]
    csv_name=result1[0]

    pool.close()
    pool.join()
    time2=time.time()-time1
    print("Time for writing all csv:%.2f seconds" %(time2))
######## create final OMI files and write the header
    for i in range(0,len(csv_name)):
        f1=open(csv_name[i],'r')
        header=f1.readline().strip('\n')
        header1=header.split(',')
        header1.insert(2,'gene_name')
        header1.append('change_type1\n')
        header1=",".join(header1)
        f2=open(filename+'_'+csv_name[i],'a')
        f2.write(header1)
```

```
        f1.close()
        f2.close()
    #cores=multiprocessing.cpu_count()-1
    pool=multiprocessing.Pool(processes=cores,maxtasksperchild=1)
######## prepare lists containing names of all reference files
    ref=[b+'_ref.txt' for b in info[0]]
    genome=[refpath+'/'+'genome_'+b+'.fa' for b in info[0]]
    pool_list=[]
######## create multiprocessing pool for parsing
    time3=time.time()
    pool_list=[pool.apply_async(vcf_func,(vcf_by_chrom[i],ref[i],
            genome[i],)) for i in range(len(info[0]))]
    result2=[c.get() for c in pool_list]
    time4=time.time()-time3
    print("Time for parsing whole vcf: %.2f seconds" %(time4))

    time5=time.time()
    if (len(csv_name) < cores):
        for j in range(0,len(csv_name)):
            pool_list=[pool.apply_async(writeOMI,(result2[i],
                    csv_name[j],filename+'_'+csv_name[j],))
                    for i in range(len(info[0]))]
    else:
        result=result2[0].copy()
        for j in range(1,len(result2)):
            result.update(result2[j])
        pool_list=[pool.apply_async(writeOMI,(result,csv_name[k],
                filename+'_'+csv_name[k],)) for k in
```

```
                    range(len(csv_name))]
    pool.close()
    pool.join()
    time6=time.time()-time5
    print("Time for writing all OMI: %.3f reconds" %(time6))
######## delete intermediate files
    for i in info[0]:
        os.remove(i+'.vcf')
        os.remove(i+'_ref.txt')

    for j in csv_name:
        os.remove(j)
    en_mem=resource.getrusage(resource.RUSAGE_SELF).ru_maxrss/1024
    times=time.time()-starting
######## delete intermediate variables to release memory
    del result1
    del result2
    del pool_list
    del pool_list1
    gc.collect()
######## record runtime information to excel file
    rexcel=open_workbook('runtime.xls')
    filelist=rexcel.sheet_by_name('sheet1').col_values(0)
    header=rexcel.sheet_by_name('sheet1').row_values(0)
    rows=rexcel.sheets()[0].nrows
    excel=copy(rexcel)
    table=excel.get_sheet(0)
    #if filename already exists, append runtime information
```

```
if(filename in filelist):
    file_index=filelist.index(filename)
    if (cores in header):
        core_index=header.index(cores)
        table.write(file_index,core_index,times)
        table.write(file_index,core_index+1,time0)
        table.write(file_index,core_index+2,time2)
        table.write(file_index,core_index+3,time4)
        table.write(file_index,core_index+4,time6)
    else:
        table.write(0,len(header),cores)
        table.write(0,len(header)+1,'generating vcf')
        table.write(0,len(header)+2,'copy csv')
        table.write(0,len(header)+3,'parsing')
        table.write(0,len(header)+4,'write OMI')
        table.write(file_index,len(header),times)
        table.write(file_index,len(header)+1,time0)
        table.write(file_index,len(header)+2,time2)
        table.write(file_index,len(header)+3,time4)
        table.write(file_index,len(header)+4,time6)
#if file does not exist, first write file information then runtime
else:
    new_row=[filename,filesize,info[2],len(info[0]),len(csv_name)]
    for i in range(len(new_row)):
        table.write(rows,i,new_row[i])
    if (cores in header):
        core_index=header.index(cores)
        table.write(rows,core_index,times)
```

```
                table.write(rows,core_index+1,time0)
                table.write(rows,core_index+2,time2)
                table.write(rows,core_index+3,time4)
                table.write(rows,core_index+4,time6)
            else:
                table.write(0,len(header),cores)
                table.write(0,len(header)+1,'splitvcf')
                table.write(0,len(header)+2,'copy csv')
                table.write(0,len(header)+3,'parsing')
                table.write(0,len(header)+4,'write OMI')
                table.write(rows,len(header),times)
                table.write(rows,len(header)+1,time0)
                table.write(rows,len(header)+2,time2)
                table.write(rows,len(header)+3,time4)
                table.write(rows,len(header)+4,time6)
        excel.save('runtime.xls')
    ######## print out filename and total time used
        print('Time elapsed: '+str(round(times,2))+' seconds')
        print('Number of cores used:%d' %(cores))
        print('VCF filename:%s' %(file))
        print(st_mem)
        print(en_mem)

import vcf
import gc

for j in range(2,11):
    vcf_reader = vcf.Reader(open('PROJECT.sorted.bam_good.vcf', 'r'))
```

```
outvcf=open('reduced'+str(j)+'.vcf','w')
my_out=vcf.Writer(outvcf,vcf_reader)
i=2
for line in vcf_reader:
    if (i % j ==0):
        my_out.write_record(line)
    i=i+1
del vcf_reader
gc.collect()
outvcf.close()
#take reduced files and remove columns one by one
for k in range(1,29):
    vcf_reader=vcf.Reader(open('reduced'+str(j)+'.vcf','r'))
    outvcf=open('reduced'+str(j)+'_'+str(k)+'.vcf','w')
    del vcf_reader.samples[-k:]
    my_out=vcf.Writer(outvcf,vcf_reader)
    for record in vcf_reader:
        my_out.write_record(record)
    del vcf_reader
    gc.collect()
    outvcf.close()
```

A.2 Extract Information from VCF File

```
import vcf
import csv
import time
import timeit
```

```
import gc

def writecsv(inputvcf):
    chr_start = time.time()
    vcf_reader = vcf.Reader(open(inputvcf,'r'))
    outfile_name=[]
    col_name=[['var_index','chrom','left','right','ref_seq','var_seq1',
    'var_seq2','count1','count2','var_score','where_in_transcript']]#,'ID'
    index=[]
    chrom=[]
    left=[]
    right=[]
    ref=[]
    alt=[]
    alt2=[]
    tags=[]
    names=locals()
    sample_names=[]
    sample_temps=[]
    base_pool=[['A'],['C'],['G'],['T']]

    record=next(vcf_reader)
    #dynamically names output file for each sample
    for i in range(0,len(record.samples)):
        names['%s'%''.join([char for char in record.samples[i].sample
                            if char.isalnum()])]=[]
        names['%s'%''.join([char for char in record.samples[i].
                            sample.lower() if char.isalnum()])]=[]
```

```
    names['count1%s' %i]=[]
    names['count2%s' %i]=[]
    sample_temps.append(''.join([char for char in record.samples[i]
                                 .sample if char.isalnum()]))
    sample_names.append(''.join([char for char in record.samples[i]
                                 .sample.lower() if char.isalnum()]))
vcf_reader = vcf.Reader(open(inputvcf,'r'))
for record in vcf_reader:
        chrom.append(record.CHROM.split())
        right.append(record.POS)
        left.append(str(record.POS).split())
        ref.append(record.REF.split())
        alt.append(record.ALT)
        tags.append(record.FORMAT.split(':'))
        for j in range(0,len(record.samples)):
            eval(sample_temps[j]).append(record.samples[j])
for m in range(0,len(tags)):
    for k in range(0,len(record.samples)):
        names['temp%s' %k]=[]
    for n in range(0,len(tags[0])):
        if(tags[0][n] not in tags[m][n]):
            for j in range(0,len(record.samples)):
                eval('temp'+str(j)).append(None)
            tags[m].insert(n,tags[0][n])
        else:
            for j in range(0,len(record.samples)):
                eval('temp'+str(j)).append(eval(sample_temps[j])[m]
```

```
[tags[0][n]])
        for j in range(0,len(record.samples)):
            eval(sample_names[j]).append(eval('temp'+str(j)))
    def get_index(input_list):
        AD=-1
        BCOUNT=-1
        for j in range(0,len(input_list)):
            if (input_list[j]=='AD'):
                AD=j
                return AD
            elif (input_list[j]=='BCOUNT'):
                BCOUNT=j
                return BCOUNT
            else:
                continue
        return
    index_count=get_index(tags[0])
    if (len(chrom)==len(left)==len(ref)==len(alt)):
            for j in range(0,len(record.samples)):
                names['list0%s' %j]=[[]]*len(chrom)
                names['temp%s' %j]=[[]]*len(chrom)
            index=[num for num in range(1,(len(chrom)+1))]
            score=[['28']]*len(chrom)
            alt2=[['']]*len(chrom)
            for i in range(0,len(chrom)):
                    if(len(alt[i])>1):
                        alt2[i]=[str(alt[i][1])]
                        alt[i]=[str(alt[i][0])]
```

```
else:
    if(ref[i][0]<str(alt[i][0])):
        alt2[i]=[str(alt[i][0])]
        alt[i]=ref[i]
    else:
        alt2[i]=ref[i]
        alt[i]=[str(alt[i][0])]

index[i]=[str(index[i])]
right[i]=[str(right[i]+len(ref[i]))]
if('BCOUNT' in tags[0]):
    pos1=base_pool.index(alt[i])
    pos2=base_pool.index(alt2[i])
    for k in range(0,len(record.samples)):
        eval('count1'+str(k)).append([eval(sample
            names[k])[i][index_count][pos1]])
        eval('count2'+str(k)).append([eval(sample
            _names[k])[i][index_count][pos2]])
elif('AD' in tags[0]):
    if(alt[i]==ref[i]):
        for k in range(0,len(record.samples)):
            eval('count1'+str(k)).append([eval
            (sample_names[k])[i][index_count][0]])
            eval('count2'+str(k)).append([eval
            (sample_names[k])[i][index_count][1]])
    else:
        for k in range(0,len(record.samples)):
            eval('count1'+str(k)).append([eval
```

```
                        (sample_names[k])[i][index_count][1]])
                        eval('count2'+str(k)).append([eval
                        (sample_names[k])[i][index_count][0]])
                else:
                    break
                for j in range(0,len(record.samples)):
                    eval('list0'+str(j))[i]= index[i]+chrom[i]+
                     left[i]+ right[i]+ ref[i]+ alt[i]+alt2[i]+
                     eval('count1'+str(j))[i]+eval('count2'+str(j))
                     [i]+score[i]#+normal[i]+ID[i]
                    eval('temp'+str(j))[i]=eval('list0'+str(j))[i]

for p in range(0,len(record.samples)):
    for ele in eval('temp'+str(p)):
        if ((ele[7]+ele[8])<=5 or ele[7]/(ele[7]+ele[8])<=0.05
            or ele[8]/(ele[7]+ele[8])<=0.05):
            eval('list0'+str(p)).remove(ele)
    index1=[num for num in range(1,(len(eval('list0'+str(p)))+1))]
    for q in range(len(eval('list0'+str(p)))):
        eval('list0'+str(p))[q][0]=index1[q]
for h in range(0,len(record.samples)):
    names['file1%s' %h]=open(sample_names[h]+'.csv',"a",newline='')
    outfile_name.append(sample_names[h]+'.csv')
    with eval('file1'+str(h)):
            writer=csv.writer(eval('file1'+str(h)))
            writer.writerows(col_name)
            writer.writerows(eval('list0'+str(h)))
for h in range(0,len(record.samples)):
```

```
            eval('file1'+str(h)).close()
        end = time.time()-chr_start
        del index
        del chrom
        del ref
        del alt
        del alt2
        for key in list(locals()):
            if (key.startswith('list0') or key.startswith('temp')):
                del locals()[key]
        del vcf_reader
        gc.collect()
        return outfile_name
```

A.3 Parse VCF File

```
from multiprocessing import Pool,Manager
import os
from math import fabs
import sys
from miscVar import AADict
import time
import csv
import gc

MAX_DIST = 5000
#function for processing vcf, returns dict
def vcf_func(vcffile,gtf,genome):
```

```
chr_start = time.time()
vcffile = open('%s'%(vcffile), 'r')
vcf = vcffile.readlines()
vcffile.close()
gtffile = gtf
f = open(gtffile,'r')
flat = f.readlines()
f.close()
seqfile=open(genome,'r')
next(seqfile)
sequence=seqfile.read().replace('\n','')
seqfile.close()

dictionary= {}
position = 0
chrom=''
genestart = 0
genestop = 0
start_time = time.time()
for l in range(0, len(vcf)): #line in vcf:
    if vcf[l].startswith('chr'):
        line = vcf[l].strip()
        vl = line.split() #split on all white space
        chrom=vl[0]
        position = int(vl[1])
        ref=vl[3]
        alt=vl[4]
        gtfCounter = -1 #point to each line in gtf file
```

```
if position not in dictionary:
    boo = 0
    where = []
    genename = []
    change_type=[]
    counter=[]
    strand_gene=[]
    while boo == 0 and gtfCounter < len(flat):
        #GSV within gene
        if position >= genestart and position <= genestop:
            counter.append(gtfCounter)
            strand_gene.append(flat[gtfCounter].split()[3])
            gtfCounter += 1
            pgstop = genestop
            #transcript start and end
            genestart = int(flat[gtfCounter].split()[4])
            genestop = int(flat[gtfCounter].split()[5])
        #Moves to the next gene if not within the range
        elif position > genestop and gtfCounter < (len(flat)-1):
            gtfCounter += 1
            pgstop = genestop #prev gene stop
            genestart = int(flat[gtfCounter].split()[4])
            genestop = int(flat[gtfCounter].split()[5])
        elif position < genestart and position > pgstop:
            boo = 1
        else:
            boo = 1
```

```
            strand = flat[gtfCounter].split()[3] #ref strand direction
            run_time = time.time() - start_time
            if gtfCounter == len(flat)-1 and position > genestop:
                #if variant position is far from the end of transcript
                if (position - genestop) > MAX_DIST:
                    genename.append('NoName')
                    where.append('Not close to gene')
                    change_type.append('')
                else:
                    #else whether close to 3' or 5'
                    genename.append(flat[gtfCounter].split()[0])
                    if strand == '-':
                        where.append('5\' untranscribed')
                        change_type.append('')
                    else:
                        where.append('3\' untranscribed')
                        change_type.append('')
            elif position < genestart and position > pgstop:
# check if closer to start or end of the one before or current
                diff1 = fabs(position-genestart) #current
                diff2 = fabs(position-pgstop) #previous
                if diff1 <= diff2 and diff1 <= MAX_DIST:
#if closer to current "new " gene and within cutoff distance
                    genename.append(flat[gtfCounter].split()[0])
                    if strand == '-':
                        where.append('3\' untranscribed')
                        change_type.append('')
                    else:
```

```
                where.append('5\' untranscribed')
                change_type.append('')
        elif diff2 <= MAX_DIST:
        #if closer to prev gene and within cutoff distance
            genename.append(flat[gtfCounter-1].split()[0])
            strand = flat[gtfCounter-1].split()[3]
            if strand == '-':
                where.append('5\' untranscribed')
                change_type.append('')
            else:
                where.append('3\' untranscribed')
                change_type.append('')
        else:
            genename.append('NoName')
            where.append('Not close to gene')
            change_type.append('')
    else:
        for i in range(len(counter)):
            genename.append(flat[counter[i]].split()[0])
            tr_start = int(flat[counter[i]].split()[4])
            tr_end = int(flat[counter[i]].split()[5])
            cd_start=int(flat[counter[i]].split()[6])
            cd_end=int(flat[counter[i]].split()[7])
            ex_starts = []
            ex_ends=[]
            x=0
            tmp = 'Unknown'
            ex_starts = [int(i) for i in flat[counter[i]].
```

```
                split()[-2].split(',')[0:-1]]
ex_ends = [int(i) for i in flat[counter[i]].
            split()[-1].split(',')[0:-1]]

while tmp == 'Unknown' and x < len(ex_starts):
    if position >= ex_starts[x] and position
       <= ex_ends[x]:
        tmp = 'Exon' #if position in exon
    elif position < ex_starts[x]:
        tmp = 'Intron'
    else:
        x+=1
if tmp == 'Intron' or tmp=='Unknown':
    where.append('Intron')
    change_type.append('')
#see if position is in translated region:
elif tmp == 'Exon':
    if position >=  cd_start and position <= cd_end:
        tmp = 'Translated'
    else: #not translated
        tmp = 'Not translated'
    if tmp == 'Not translated':
        if cd_start == cd_end: #ncRNAs
            where.append('Not translated, ncRNA')
            change_type.append('')
        else: #utrs
            if position  < cd_start:
                if strand_gene[i] == '-':
```

```
                                        where.append('Not translated
                                    5\' utr')
                                        change_type.append('')
                                else:
                                        where.append('Not translated,
                                    3\' utr')
                                        change_type.append('')
                            else:
                                if strand_gene[i] == '-':
                                        where.append('Not translated,
                                    3\' utr')
                                        change_type.append('')
                                else:
                                        where.append('Not translated,
                                    5\' utr')
                                        change_type.append('')
                        elif tmp == 'Translated':
                            where.append('CDS')
                            ref_codon=''
                            var_codon=''
                            m=0
                            m = (position - cd_start) % 3
                            if len(ref)==len(alt)and position<
                                            len(sequence)-2:
                                if m==0 :
                                    ref_codon=ref+sequence[position+1]+
                                        sequence[position+2]
                                    var_codon=alt+sequence[position+1]+
```

```
            sequence[position+2]
        elif m==1:
            ref_codon = sequence[position-1]
                +ref+ sequence[position+1]
            var_codon = sequence[position - 1]
                +alt+sequence[position+1]
        else:
            ref_codon = sequence[position-2]+
                sequence[position-1]+ref
            var_codon = sequence[position-2]+
                sequence[position-1]+alt
        if strand_gene[i] == '-':
            intab = "ATCG"
            outtab = "TAGC"
            trantab = str.maketrans(intab,
                                    outtab)
            var_codon = var_codon.translate(
                trantab)
            ref_codon = ref_codon.translate(
                trantab)
        ref_aa=''
        var_aa=''
        if var_codon in AADict:
            var_aa = AADict[var_codon]
        if ref_codon in AADict:
            ref_aa = AADict[ref_codon]
        if var_aa == ref_aa:
            change_type.append('Synonymous')
```

```
                                    else:
                                        change_type.append('Non-synonymous')
                                else:
                                    change_type.append('Non-synonymous')
                            else:
                        #To make sure to fill change type in on the ignored ones.
                                where.append('Ignore')
                                change_type.append('')
                        else:
                            where.append('Ignore')
                            change_type.append('')
                    dictionary[position]=[chrom,genename,where,change_type]
        chr_end =  time.time() - chr_start
        del vcffile
        del seqfile
        gc.collect()
        return dictionary
```

A.4 Codon Dictionary

```
AADict = {"TTT":"F", "TTC":"F", "TTA":"L", "TTG":"L",
    "TCT":"S", "TCC":"S", "TCA":"S", "TCG":"S",
    "TAT":"Y", "TAC":"Y", "TAA":"*", "TAG":"*",
    "TGT":"C", "TGC":"C", "TGA":"+", "TGG":"W",
    "CTT":"L", "CTC":"L", "CTA":"L", "CTG":"L",
    "CCT":"P", "CCC":"P", "CCA":"P", "CCG":"P",
    "CAT":"H", "CAC":"H", "CAA":"Q", "CAG":"Q",
    "CGT":"R", "CGC":"R", "CGA":"R", "CGG":"R",
```

```
"ATT" : "I" , "ATC" : "I" , "ATA" : "I" , "ATG" : "M" ,
"ACT" : "T" , "ACC" : "T" , "ACA" : "T" , "ACG" : "T" ,
"AAT" : "N" , "AAC" : "N" , "AAA" : "K" , "AAG" : "K" ,
"AGT" : "S" , "AGC" : "S" , "AGA" : "R" , "AGG" : "R" ,
"GTT" : "V" , "GTC" : "V" , "GTA" : "V" , "GTG" : "V" ,
"GCT" : "A" , "GCC" : "A" , "GCA" : "A" , "GCG" : "A" ,
"GAT" : "D" , "GAC" : "D" , "GAA" : "E" , "GAG" : "E" ,
"GGT" : "G" , "GGC" : "G" , "GGA" : "G" , "GGG" : "G" }
```

Curriculum Vitae

Bofei Wang was born on March 27, 1993 in China. Being the only child in the family, his parents took great effort in educating him and helping him build career goals. He entered Tianjin University of Science and Technology, Tianjin, China in 2011 and graduated in 2015. Then he started his studies at Bioinformatics Program at The University of Texas at El Paso (UTEP). In 2017, he got his master degree and joined Computation Science Program at UTEP. During this period, he has been working as a Teaching Assistant in Mathematical Sciences Department. In 2018, he was elected as the president of Chinese Student and Scholar Association.