AN ABSTRACT OF THE PAPER OF

Jun He for the degree of Master of Science in Computer Science presented on November 23, 2016.

Title: Variant Annotation

Abstract approved: _____

Stephen Ramsey

With the development of technologies in genome sequencing and variant detection, a huge number of variants are detected. To further analyze the variants, it requires an efficient tool to annotate the functional effect of variants. This project managed to develop an efficient program to annotate the functional effect of variants based the locations of variants. It also proposed two algorithms, an array-based algorithm and a tree-based algorithm, to expedite the searching process. The array-based algorithm is more efficient for sorted variants whereas the tree-based algorithm is more efficient for unsorted variants. Compared with naive searching algorithm, the array-based algorithm reduces the searching complexity from $O(NK)$ to $O(N + K)$ while the tree-based algorithm reduces to O(K log N), where N is the number of genes and K is the number of variants. According to the experiment, the two algorithms succeed in annotating the variants and take less than 1 second to annotate 10^5 variants whereas naive algorithm takes about 133 seconds.

Variant Annotation

by

Jun He

A PAPER

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented November 23, 2016
Commencement June 2017

Master of Science paper of Jun He presented on November 23, 2016.

APPROVED:

Major Professor, representing Computer Science

Director of the School of Electrical Engineering and Computer Science

---

Dean of the Graduate School

I understand that my paper will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my paper to any reader upon request.

Jun He, Author

ACKNOWLEDGEMENTS

I would like to thank Professor Stephen Ramsey for his insightful advice in this project. He also introduce me to the research area of Bioinformatics and help me a lot with my job search. I believe that all I have learned from him would benefit my future career.

I would also like to thank all other committee members, Prof. Prasad Tadepalli, Prof. Molly Megraw for their insights, suggestions and their time for my final exam.

Thanks to all my colleagues and friends for their support and help, special thanks to Tanjin Xu, Xin Liu, Yao Yao and Zheng Liu putting up with me throughout my graduate life in U.S..

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ALGORITHMS

Chapter 1: Introduction

1.1 Introduction to the DNA Variant Annotation

DNA, i.e. deoxyribonucleic acid, is the molecular basis of heredity in humans and almost all other organisms. Nearly every cell in an organism has the same DNA. Most DNA is located in the cell nucleus (where it is called nuclear DNA), while a small portion of DNA can also be found in the mitochondria (where it is called mitochondrial DNA or mtDNA). DNA is a polymer made up of a sugar-phosphate backbone and four nitrogenous chemical bases: adenine (A), guanine (G), cytosine (C), and thymine (T) [1]. Human DNA consists of about 3 billion bases and is approximately 99.9% identical from person to person. The sequence of these bases stores the information for building and maintaining the organism.

A gene is the basic physical and functional unit of heredity. Genes, which are made up of DNA, contain instructions for making proteins, through trinucleotide sequences called codons. In humans, genes vary in size from a few hundred DNA bases to more than 2 million bases. A gene consists of a consecutive sequence of DNA. However, not all of the sequence is converted into mature messenger RNA (mRNA) to code for proteins. There are two main components inside a gene, introns and exons. Introns are the parts of a gene that do not code for the protein amino acid sequence, whereas exons contain the codons that describe the protein's amino acid sequence. Introns can range in size from tens of base pairs to hundreds of thousands of base pairs [3]. An example diagram of a gene is shown in Figure 1.1. The whole line is a chromosome. The green parts indicate genes. We can see two genes reside in the chromosome. Inside the gene, the red parts are exons. The green areas indicate introns. The Human Genome Project has estimated that humans have between 20,000 and 25,000 genes [2], and of these genes, different subsets of genes will be expressed in different cells at different times.

In order for a cell to divide, it must first replicate its DNA. In most cases, replication is accurate, but occasionally, an error can occur, leading to a specific localized difference in the DNA content of the progeny cells. Such a change is called a mutation, and it can arise in either germinal cells (in which case it is called a germline mutation, which can be passed on to future generations) or in non-germinal cells (in which case it is called a somatic mutation). The general

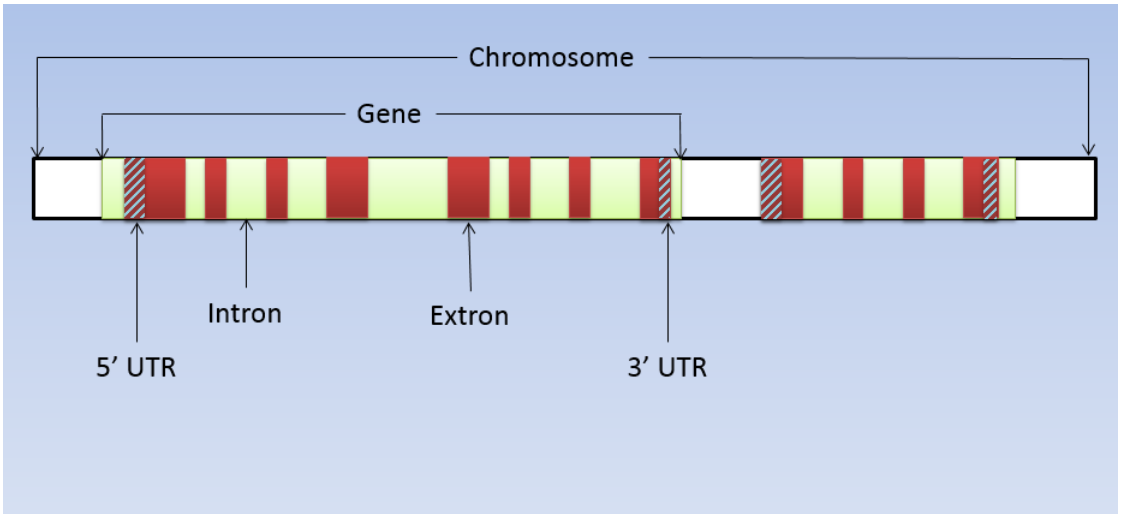

Figure 1.1: Diagram of a gene, showing different sub-regions of the gene in which a variant could be located.

term for a DNA alteration–whether it is rare or common in the population, or restricted to somatic cells–is a *variant.*

Most DNA variants reside in the non-coding portion of the genome, and thus do not change the polypeptide sequences produced by genes. However, when a variant is located in the protein-coding portion of a gene, it can in some cases lead to changes in the protein's polypeptide sequence and/or prevent expression of the protein altogether. In somatic cells, when a mutation increases the propensity of a cell to undergo uncontrolled cell division, it can lead to cancer, in which case the mutation would be under *positive selection* in the cancer.

With the development of high-throughput genome sequencing technology, variants can be detected genome-wide. However, in the case of cancer, the number of somatic variants that may be detected in a cancer can number in the tens of thousands, and thus, a computational tool is needed that can help prioritize variants by their likelihood to have an effect on protein function.

1.1.1 Functional Effects of Variants

The aim of this project was to develop a software tool that can annotate variants for potential functional effects, based on the location where the variant occurs within a gene. The secondary aim was to explore the running time and memory usage for different implementations of the core

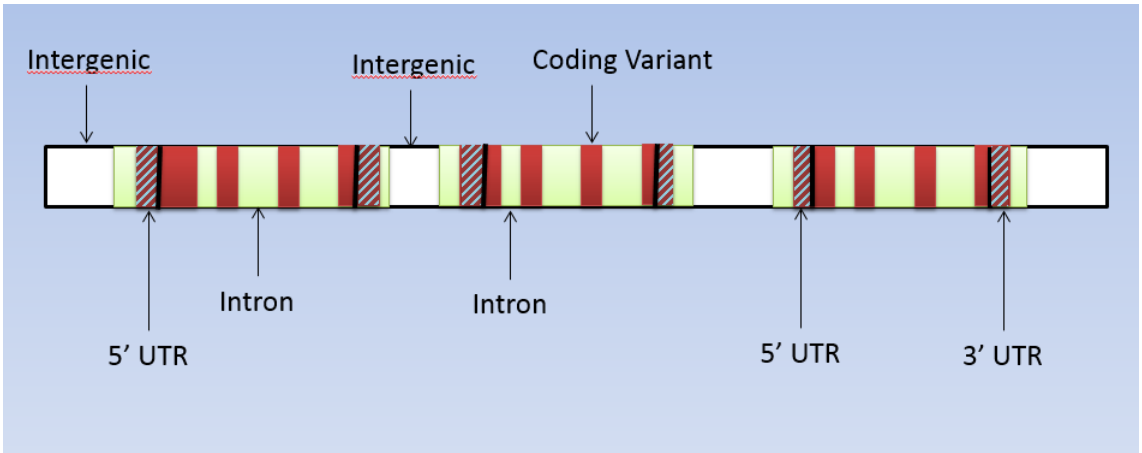

Figure 1.2: potential variant functional regions

algorithm for variant annotation. We defined seven potential functional effects. The corresponding functional regions are shown in Figure 1.2.

- Intergenic variant
  An intergenic variant does not reside in any genes, and thus it does not alter any amino acid sequences. Therefore, its putative impact is low.

- Intron variant
  An intronic variant resides in a gene, but not in an exon. Thus, it does not alter any amino acid. Its putative impact is still low but a little higher than Intergenic variant, because of the possibility that it could impact RNA splicing.

- 5′ untranslated region (UTR) variant
  A 5′ UTR variant resides in an exon, but is prior to the start of coding sequence. Although it still does not alter any amino acid sequences, it is relatively more damaging than intergenic and intron variant.

- 3′ UTR variant
  A 3′ UTR variant resides in an exon, but after the stop codon of the open reading frame. The putative impact is similar to 5′ UTR variant.

- Non-coding exon variant
  A non-coding exon variant resides in an exon. But the corresponding gene has no start or

stop codon, resulting in no coding sequences. Thus, no amino acid sequences would be altered.

- Synonymous variant
  A synonymous variant resides in an exon and coding sequences, leading to altering a coding sequence. However, since several DNA codons correspond to one amino acid, the resulting amino acid sequence does not change.

- Non-synonymous variant
  A non-synonymous variant resides in coding sequences and alters an amino acid sequence. Its putative impact is high.

The following table shows the notation and putative impact for each variant effect [5].

Variant effect	Putative impact
intergenic_region	MODIFIER
intron_variant	MODIFIER
5_prime_UTR_variant	MODIFIER
3_prime_UTR_variant	MODIFIER
non_coding_exon_variant	MODIFIER
synonymous_variant	LOW
non_synonymous_variant	MODERATE

For a non-synonymous variant, we annotate the reference amino acid, alternate amino acid and the position of the amino acid instead of the term.

Chapter 2: Methodology

2.1 Input

2.1.1 Gene structure data

The gene structure data file, in Gene Transfer Format (GTF, [7]), contains information about gene structure, such as the start codon, stop codon, exon, strand and gene ID. This is the key data to determine the functional region of a given variant. An example exerpt from a GTF file is shown in Figure 2.1.

```
chr1    canFam3_ensGene exon    8350657 8351219 0.000000        +       .       gene_id "ENSCAFT00000047983";
chr1    canFam3_ensGene exon    8359233 8359447 0.000000        +       .       gene_id "ENSCAFT00000047983";
chr1    canFam3_ensGene start_codon     8365671 8365673 0.000000        +       .       gene_id "ENSCAFT000000
chr1    canFam3_ensGene CDS     8365671 8365716 0.000000        +       0       gene_id "ENSCAFT00000047983";
chr1    canFam3_ensGene exon    8365645 8365716 0.000000        +       .       gene_id "ENSCAFT00000047983";
chr1    canFam3_ensGene CDS     8366024 8366362 0.000000        +       2       gene_id "ENSCAFT00000047983";
chr1    canFam3_ensGene exon    8366024 8366362 0.000000        +       .       gene_id "ENSCAFT00000047983";
chr1    canFam3_ensGene CDS     8410380 8410727 0.000000        +       2       gene_id "ENSCAFT00000047983";
chr1    canFam3_ensGene exon    8410380 8410727 0.000000        +       .       gene_id "ENSCAFT00000047983";
chr1    canFam3_ensGene CDS     8427874 8427976 0.000000        +       2       gene_id "ENSCAFT00000047983";
chr1    canFam3_ensGene exon    8427874 8427976 0.000000        +       .       gene_id "ENSCAFT00000047983";
chr1    canFam3_ensGene CDS     8453827 8453842 0.000000        +       1       gene_id "ENSCAFT00000047983";
```

Figure 2.1: GTF data example

2.1.2 Genome data

For the purposes of variant annotation, the reference genome consists of the complete nucleotide sequences of each chromosome. The reference genome is often stored in a faidx-indexed reference file in the FASTA format [6]. We will use this file to get the amino acid codon of a given variant. An example of the first 350 bp of chromosome 1 of the reference genome assembly of *Canis familiaris* (the domestic dog, assembly version CanFam3.1) is shown in Figure 2.2. In this project, we used somatic variant calls from exome sequencing of canine bladder cancer samples and matched blood samples, in VCF format.

```
>1
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNTATGTGAGAAGATAGCTGAA
CGCCTTGTCCACATCATCTTACTGCTGAGAGTTGAGCTCACCCTCAGTCCCTCACAGTTC
CACACTGCCTGCAGAGTGAGTTTCCCATGTCTTCACCAGAGACTTTTGCCAGAGGCTTCT
GAGACGCAAGTTAACAATGCAGACCTGGAGGGTATCTCCAGGTGCAGTAGAGTGGTAATC
TCGGAACCTCCTGACTCAGAATACTGCTACCTTCACACTGTCATAAGAATGCAGCGAGTT
GAGAGCTGGCTTCTAGGCATGCTTCCTTTTGAGAGCTGAGGACAGGACAGAACCCTCCCG
```

Figure 2.2: Genome data example

2.1.3 Variant data

This is the data to be annotated. The key information includes the chromosome, position, reference base and alternate variant alleles of the variant. The data are typically stored in a variant call format (VCF [4]) file. We will annotate the variant by appending the annotation information to the INFO field [5] in the VCF file. An example excerpt from a VCF file is shown in Figure 2.3.

```
#CHROM  POS     ID      REF     ALT     QUAL    FILTER  INFO    FORMAT  C       sample1
1       366459  .       G       T       .       PASS    SC=0;KS=NOVEL;CV=COVERED;PW=0.99993;TP=0.9
1;n_ref_sum=40;n_alt_sum=KEEP       RC:AC:RS:AS     1.954141:91:4:3398      0:GG:23.515183:92
1       724795  .       G       T       .       PASS    SC=0;KS=NOVEL;CV=COVERED;PW=0.999218;TP=0.
n_ref_sum=0;n_alt_sum=KEEP RC:AC:RS:AS     2.1856:76:4:2866        0:GG:24.594482:82
1       1244743 .       C       A       .       PASS    SC=0;KS=NOVEL;CV=COVERED;PW=0.999956;TP=0.
=3163;n_ref_sum=0;n_alt_sum=KEEP   RC:AC:RS:AS     2.027738:94:4:3575      0:CC:24.982382:83
1       1244775 .       G       A       .       PASS    SC=0;KS=NOVEL;CV=COVERED;PW=0.999835;TP=0.
```

Figure 2.3: Variant data example

2.2 Output

Because a gene can be transcribed into multiple transcript isoforms, one variant may have several possible effects corresponding to different transcripts. For each variant, our annotation software produces a list of potential effects. The potential effects are listed in section 1.1. In the project, the output is stored in a VCF file in the INFO field. To be specific, a new subfield, named ANN, will be appended to INFO field [5]. The most important information is the effect of variants and the putative impact of the effect.

Besides the effect of variants, the corresponding Ensembl transcript id and common gene name are also annotated. All the annotation formats follow the standard variant annotation in

VCF format[4]. An example of annotation information added to the INFO field of a VCF file record is shown in Figure 2.4.

```
ANN=A|intron_variant|MODIFIER|ENSCAFT00000046825|ENSCAFG00000000012|ATP9B,
```

Figure 2.4: Annotation example

2.3 Algorithms

There are tree datasets to be kept track of. If we search on the whole datasets for each variants, the total running time complexity is $O(mnk)$, where m, n and k is the size of each file. In addition, the size of the datasets may be too large to load them all into RAM. For instance, the size of a typical human genome file is about 3.0 GB. To solve the difficulties in this problem, we develop several algorithms to annotate variants.

The whole problem can be split into two smaller subproblems. The first one is to obtain a list of genes in which a variant resides. The second one is to annotate each functional region. And if the variant resides in a coding sequence, get the corresponding DNA sequence around the variant and translate it into amino acids.

The most challenging part is to get the gene list since genes may overlap. The naive solution is to search on the whole genome for each gene, resulting in a time complexity of $O(NK)$, where N is the number of genes and K is the number of variants. To speed up the annotation, two algorithms were proposed. One algorithm is array based and efficient for sorted variants while the other is tree based and efficient for unsorted variants.

2.3.1 Outline of the method

The outline annotation method can be summarized in 3 steps. 1. Get a variant. 2. Search for genes in which the variant resides. 3. Annotate the variant with respect to each gene. The following algorithm describe the complete annotation method in detail.

2.3.2 Searching algorithm

Algorithm 2.3.1 gives a detailed description of the annotation process. However, a critical step in algorithm 2.3.1 is still undefined. This section discusses 3 algorithms to search the gene list.

Algorithm 1 VARIANT ANNOTATION

input : gene structure data G. reference genome data R. variant data V.
output: The annotated variant data V

for *each variant v in V at chromosome chr and position pos* **do**
  $gene\_list$ = get_gene_list(G, chr, pos)
  $A = \varnothing$
  if *gene_list is empty* **then**
    A = A + "intergenic_region"
  end
  for *each gene in gene_list* **do**
    if *gene has neither start codon nor stop codon* **then**
      A = A + "non_coding_exon_variant"
    end
    else if *v does not reside in any exon of gene* **then**
      A = A + "intron_variant"
    end
    else if *v resides prior to start codon of gene* **then**
      A = A + "5˙prime_UTR_variant"
    end
    else if *v resides after stop codon of gene* **then**
      A = A + "3˙prime_UTR_variant"
    end
    else
      get reference amino acid ra and alternative amino acid aa
      if ra == aa **then**
        A = A + "synonymous_variant"
      end
      else
        A = A + "ra + pos + aa"
      end
    end
  end
  Annotate v with A
end

return The annotated variants V

The first algorithm is a naive search algorithm that works intuitively. The other two algorithms are proposed to expedite the search process. One is array based and most efficient for sorted

variants while the other is tree based and most efficient for unsorted variants.

2.3.2.1 Naive searching

The idea of naive searching is relatively simple: start searching for each variant from the first gene. In this way, the average time complexity for one variant is $O(N)$, where N is the number of genes. Thus, the total time complexity is $O(NK)$, where K is the number of variants.

Algorithm 2 NAIVE SEARCHING

input : gene structure data G, chromosome chr and position pos
output: a list of genes which reside in chr and overlap pos

```
get gene array gene_arr
i = 0
gene_list = ∅
while i < length of gene_arr and pos > the start of gene_arr[i] do
    if pos < the end of gene_arr[i] then
        add gene_arr[j] to gene_list
        i = i + 1
    end
end
return gene_list
```

2.3.2.2 Array based searching

An array based search algorithm is most efficient for sorted variants. Genes are stored in sorted order in an array. While iterating though the variants, the algorithm keeps track of the last visited gene. Each variant starts searching from the last visited gene, not from the beginning. At the end, all of the variants and genes are visited only once, resulting in a complexity of $O(N + K)$, where N is the number of genes and K is the number of variants. The memory complexity is $O(N)$, with a few extra memory cost for maintaining the array data structure.

Algorithm 3 ARRAY BASED SEARCHING

input : gene structure data G, chromosome chr and position pos
output: a list of genes which reside in chr and overlap pos

get gene array $gene\_arr$ and last visited index i from G on chromosome chr

while $i <$ *length of* $gene\_arr$ *and pos* $>$ *the end of* $gene\_arr[i]$ **do**
    $i = i + 1$
end

update last visited index i to G
$gene\_list = \varnothing$
$j = i$

while $j <$ *length of* $gene\_arr$ *and* $gene\_arr[j]$ *overlap* pos **do**
    add $gene\_arr[j]$ to $gene\_list$
    $j = j + 1$
end

return $gene\_list$

2.3.2.3 Tree based searching

The tree based algorithm is most efficient for unsorted variants. In this algorithm, genes are stored in an interval tree [8]. Although each variant searching on the whole gene, it takes only $O(logN)$ time [8] to get the gene list. Thus, the total time complexity is $O(K(\log N))$. If sort the variants and apply array based algorithm, the total time complexity would be $O(K(\log K)+N)$, which is larger than $O(K(\log N))$. Memory complexity is still $O(N)$, but with more extra memory cost for maintaining the tree structure.

An interval tree is very similar to a binary search tree. A standard binary search tree usually maintains a single value for each node while interval tree maintains an interval for each node. When searching genes, it makes a preorder traversal on the interval tree until the position is completely outside of the current node.

The key to ensuring efficient search in a binary search tree is ensuring that the three is balanced. Since genes rarely change, we don't need a balanced binary search tree algorithm here, such as Red Black tree. We could sort the genes by their starting position first and then build a balanced binary search tree with it.

Algorithm 4 TREE BASED SEARCHING

input : gene structure data G, chromosome chr and position pos
output: a list of genes which reside in chr and overlap pos

Function *get_gene_list(G, chr, pos)*
    get gene tree root $gene\_root$ from G on chromosome chr
    $gene\_list = \varnothing$
    $binary\_search(gene\_root, pos, gene\_list)$
    return $gene\_list$

Function *$binary\_search$(root, pos, gene_list)*
    if *$root$ is empty* **then**
        return
    end
    if *pos resides outside region of $root$* **then**
        return
    end
    g = the gene in $root$
    if *pos resides in g* **then**
        add g to $gene\_list$
    end
    $binary\_search(root.leftchild, pos, gene\_list)$
    $binary\_search(root.rightchild, pos, gene\_list)$

2.4 Other issues

2.4.1 how to get amino acid

The whole genome sequence data is very large. For instance, the size of a typical human genome file is about 3.0 GB. However, since coding sequences amount to only 1.5% of whole genome, we could preload the coding sequence to the gene structure data. In this project, the final gene structure data is about 100 MB, which is small enough to load in most machines' memory.

2.4.2 strand

Another problem is that genes have different strand. Different from positive strand, negative strand starts from a larger chromosomal coordinate and ends at a smaller chromosomal coordinate. In addition, the nucleotide sequence must be reversed according to the format A-T, C-G.

Chapter 3: Experiment Result

3.1 Success rate

Both array based algorithm and tree based algorithm have a 100% success rate on the manually annotated test data. The following are several typical annotation example in detail.

3.1.1 Intergenic Variant

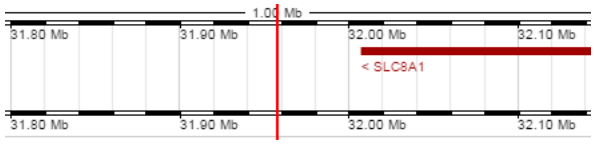

Figure 3.1: Intergenic Variant

Figure 3.1 illustrate an example for annotation of an intergenic variant. The variant is located on canine chromosome 17 at coordinate 31957825. The red vertical line denotes the location of the variant. We can see that this variant does not reside on any genes. The closest gene is *SLCSA1*.

3.1.2 Intron Variant

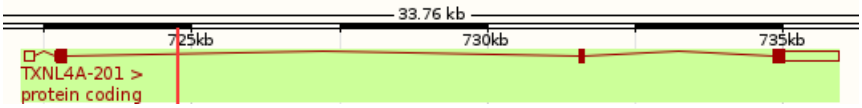

Figure 3.2: Intron Variant

Figure 3.2 illustrate an example for intron variant. The variant is located on canine chromosome 1 at coordinate 724795. The red vertical line denotes the location of the variant. We can see that this variant resides on gene *TXNL4A* (Ensembl transcript 201). The red small solid

rectangles denote exons in the gene whereas the hollow rectangles denote the 5′ UTR and 3′ UTR. We can see that the variant does not reside on any exons. Thus this variant is annotated as intron variant.

3.1.3 Synonymous Variant

Figure 3.3: Synonymous Variant

Figure 3.3 shows an example for synonymous variant. The variant is located on canine chromosome 1 at coordinate 4838585 and it alters C to A. The red hollow rectangle denotes the location of the variant. The first line is the amino acid sequence whereas the second line is the DNA sequence. Note that the gene in which the variant resides is on the reverse strand of the chromosome so that both the two sequence should be read from right to left, The original amino acid at this position was G (glycine), which is made up of DNA sequence GGC. After the variant took effect, alternate DNA sequence became GGA. According to the DNA codon table, the resulting amino acid is still G (glycine). The variant does not change the amino acid. Thus this variant is annotated as synonymous variant.

3.1.4 Non-synonymous Variant

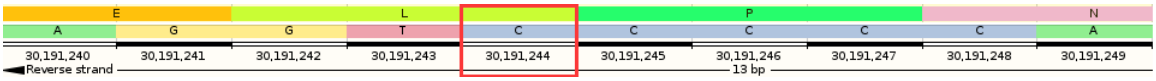

Figure 3.4: Non-synonymous Variant

Figure 3.4 is an example of a nonsynonymous variant. The variant is located on canine chromosome 2 at coordinate 30191244 and it alters C to A. The red hollow rectangle denotes the location of the variant. The first line is the amino acid sequence whereas the second line is the DNA sequence. Note that the gene in which the variant resides is on the reverse strand of the chromosome so that both the two sequence should be read from right to left, The original amino acid at this position was L (leucine), which is made up of DNA sequence CTG. After the

variant took effect, alternate DNA sequence became ATG, leading to a different amino acid M (methionine). The variant changes the amino acid and may affect the function of the protein. Thus this variant is annotated as non-synonymous variant in the form L1026M, where 1026 is the location of the amino acid in the whole protein.

3.1.5 Variant on Multiple Genes

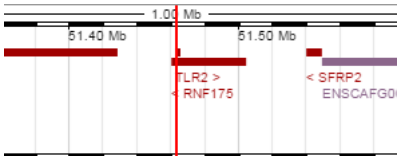

Figure 3.5: Variant on Multiple Gene

Figure 3.5 illustrate a special example of a variant that resides in multiple genes. The variant is located on canine chromosome 15 at coordinate 51463688 and it alters G to T. We can see that *TLR2* and *RNF175* overlap and the variant resides in the two genes. This is a rare case and our algorithm detect the 2 genes and annotated them correctly.

3.2 Running time

3.2.1 Sorted variants

The following table shows the running time (in seconds) for annotation of pre-sorted variants. There are 10^5 variants in this experiment.

sample size	Naive	Array	Tree
10^5	132.7	0.3751	0.6567

We can see that both the array-based algorithm and the tree-based algorithm significantly reduce the running time from 132.7 seconds to less than 1 second. Since variants are already sorted, the array-based algorithm is even more efficient than tree based algorithm.

3.2.2 Unsorted variants

The following table shows the running time for unsorted variants in seconds. Variants are sorted first to make the array based algorithm work for this experiment. The time of sorting variants is included. There are 10^5 variants in this experiment.

sample size	Naive	Array	Tree
10^5	133.76	0.8034	0.724

We can see that the array-based algorithm and the tree-based algorithm are still much more efficient than naive searching. Since naive searching and the tree-based algorithm do not care about the order of variants, the running time in this experiment for those 2 algorithms is similar to the sorted-variants experiment. However, the array-based algorithm requires an extra time to sort the variants, leading to a dramatic increase in time.

3.3 Memory cost

The following table shows the memory cost each algorithm in megabytes. As discussed above, all three algorithms have the same memory complexity but with different overhead. The result demonstrate the analysis that tree based algorithm consumes the most memory whereas naive algorithm takes the least.

sample size	Naive	Array	Tree
10^5	649.5 MB	653.8 MB	675.7 MB

Bibliography

[1] Saenger, Wolfram, *Principles of Nucleic Acid Structure*, Springer-Verlag, New York, 1984.

[2] *The Human Genome Project (HGP)*, https://www.genome.gov/10001772/ 2003.

[3] Alberts B, Johnson A, Lewis J, Raff M, Roberts K, Walter P, *Molecular Biology of the Cell*, Garland Science, New York, 2002.

[4] *The Variant Call Format (VCF)Specification*, https://github.com/samtools/hts-specs, Version 4.2, 2015.

[5] Pablo Cingolani, Fiona Cunningham, Will McLaren, Kai Wang *Variant annotations in VCF format*, Version 1.0 2015.

[6] *FASTA format*, https://en.wikipedia.org/wiki/FASTA˙format 2016.

[7] *GTF format*, http://www.ensembl.org/info/website/upload/gff.html 2016.

[8] Charles E. Leiserson, Thomas H. Cormen, Clifford Stein, Ronald Rivest *Introduction to Algorithms*, MIT Press, Massachusetts, 3rd edition, 2009.