Vcall

Group Members:
- Jiayu Chen
- Kaifu Yang

Github Repository: https://github.com/JohnChen034/CSE185_project

Brief Introduction:

By leveraging the power of the pandas library in Python, we have developed a tool called Vcall that filters and transforms raw sequence data from a Mpileup file into a VCF (Variant Call Format) file. This conversion enables researchers to easily identify and analyze potential SNPs changes in the dataset.

The goal of our tool is to facilitate the analysis of genomic data by providing a streamlined process for identifying and interpreting single nucleotide polymorphisms (SNPs) in sequencing data. SNPs are variations in a single nucleotide at a specific position in the genome and can have significant implications in understanding genetic variations, disease associations, and population genetics. For example, researchers studying the genetic basis of diseases can use our tool to identify potential disease-associated SNPs. By comparing the genomic data of affected individuals with healthy controls, they can identify genetic variants that may contribute to disease susceptibility or progression.

Methods:

To achieve this, we implemented various functions and parameters inspired by Varscan, a popular variant calling tool. We carefully selected and integrated seven parameters: minimum frequency threshold for variants, the minimum read depth required to make a call, the minimum supporting reads to confidently identify variants, the minimum base quality to count a read, the minimum frequency to classify a variant as homozygous, the p-value threshold for calling variants, and the option to ignore variants with strong bias towards one strand.

The implementation follows a strict order of steps. Firstly, we applied three basic filters to the original reads. We eliminated special characters (e.g., '^') denoting the start and end positions of a read, as they are irrelevant to SNP calling. We focused on positions with at least one variant read, discarding positions with no variants present to enhance computational efficiency. Additionally, we disregarded positions with insertions or deletions, as they hold no significance in SNP calling. During this stage, we also calculated the FORMAT column in the VCF file, preparing for subsequent filtering processes.

In the second phase, we implemented advanced filters, which are divided into two layers that align with the command line arguments. The order of these filters is crucial for their effectiveness. We began by filtering based on the minimum read depth at a position, filtering out any read that does not have enough coverage at its position. Subsequently, we applied a filter based on the base quality of each read, ensuring that only reads above a certain threshold were considered. Next, we implemented filters based on the minimum variant frequency and minimum variant supporting reads, allowing us to identify positions with variant numbers exceeding an actual integer threshold and a percentage based on the coverage of the position. For such positions, we further filtered them based on their p-values, with values smaller than threshold indicating a significant difference between the variant number and the normal number within the given context. Finally, we analyzed the distribution of variants across each strand and ignored variants with more than 90% support on a single strand to exclude possible false positive results.

By carefully designing and implementing these steps, our tool provides an efficient and accurate means of filtering and identifying potential SNPs in the input Mpileup file. Its seamless integration of pandas and adherence to the parameter specifications of Varscan ensures reliable results for researchers in the field.

We benchmark against Varscan and we are benchmarking on the lab1 data including the NA12878_child, NA12892_mother, and NA12891_father combined mpileup file extracted in chr6:128405804-128605805. We use this mpileup file as input for Varscan and Vcall (ours) for the benchmarking experiment.

We input --min-var-frequency 0.2 --min-freq-for-hom 0.8 --p-value 0.01 --min-reads 2 --min-avg-qual 15 --strand-filter True -min-coverage 8 –variants for parameters. Figure 2 shows that Vcall uses about 35 seconds while Varscan uses 15 seconds in runtime. Other than that, Vcall produces identical results as Varscan, for all output values including chromosome number, position, p-value, and other informational statistics for that position. Compare to what we had on presentation, we solved the issue with strand filter. This decreases our speed from 40 seconds to 35 seconds and increase out accuracy from 96% (2 False positive) to 100% on the benchmark dataset.

Figure 2

	time	total variant
Vcall	35.954s	41
Varscan	15.299s	41

Figure 3

		Varscan	
		TRUE	FALSE
Vcall	TRUE	41	0
	FALSE	0	

We also tested on a online database containing the genomic sequence of Saccharomyces cerevisiae in fastq format. We generate a mpileup file use commands from Lab1. The results are shown in Figure 4.1, 4.2, 4.3. In summary, have 5-6 times the speed of Varscan and identified less genes than Varscan. However, these False Positives positions are mostly in this format (Figure 4.4), which is meaningless since it includes no reads in that position. Other than these positions, we have identical statistical information for the identical positions and good accuracy rate benchmarking to Varscan.

Figure 4.1

Test1		Varscan (39s)	
		TRUE	FALSE
Vcall (3m 48s)	TRUE	113	18
	FALSE	8	

Figure 4.2

Test2		Varscan (26s)	
		TRUE	FALSE
Vcall (3m 28s)	TRUE	112	17
	FALSE	3	

Figure 4.3

Test3		Varscan (38s)	
		TRUE	FALSE
Vcall (3m 31s)	TRUE	109	33
	FALSE	5	

Figure 4.4

We perform most of the statistics in the VCF by python's default functions, including count and regular expressions. We substitute some that uses pandas functions into using

numpy because it increases performance efficiencies. In p-value calculation on variants association, we use Fisher Exact test function from scipy.stats tool to generate pvalue. Here is a list of version number we use for our tool: Python version: 3.9.5, pandas version: 1.5.3, re version: 2.2.1, argparse version: 1.1, math module version: sys.version_info(major=3, minor=9, micro=5, releaselevel='final', serial=0), numpy version: 1.21.1, scipy version: 1.7.0.

For our real dataset, we used data from Genome-in-a-bottle. We downloaded the data of the Chinese trio and indexed them through samtools. Then, we mpileup the first two million reads of chromosome I of the three samples–son, mother, and father–together using samtools. Then we run our tools on it with flags: --min-var-frequency 0.2 --min-freq-for-hom 0.8 --p-value 0.01 to output the result.

Results:

There are in total 16541 positions detected in the vcf file. We then use the pyvcf library to find the positions with mutation, in which child has alternative alleles while parents do not. 1134 mutations are detected. The total number of mutations is too large so we pick SNPs that belongs to two regions to interpret. Many of the other mutations are in intron or do not gather in one gene's exon so we did not investigate further.

First, at the end of the reads we mpileuped, from chr1:1866241 to chr1:1913716 (Figure 5.1 and 5.2), there are in total 24 mutations we detected that are in the exon of gene CFAP74. This gene may play a role in cilium movement(according to GeneCards). According to two papers, the mutations in this gene could be related to a disease: Primary Ciliary Dyskinesia(PCD).[Sha, Y][Biebach, L] However, these mutations could not indicate the conditions of the sample and we do not know the condition of the sample. We are just proposing interesting findings in the trend of the mutations in our output.

Moreover, from chr1:1034569 to chr1:1044482 (Figure 5.3 and 5.4), there are 18 mutations on the exon of the gene C1orf159. This gene is found to be an unfavorable prognosis marker for renal and liver cancer, and a favorable prognosis marker for urothelial cancer[From website proteinatlas]. The mutations could potentially make a difference in its protein structure thus its expression.

Figure 5.1

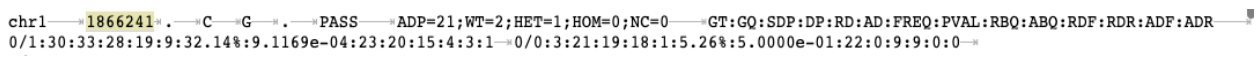


Figure 5.2

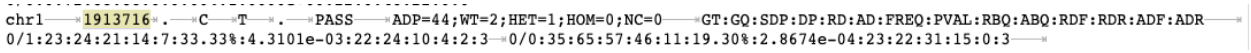


Figure 5.3

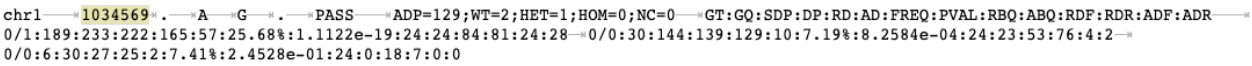
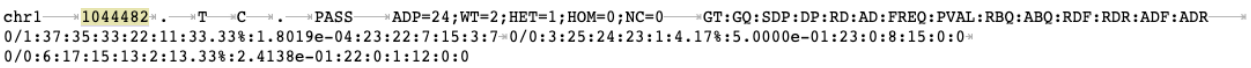


Figure 5.4



Discussion:

We are currently facing some challenges related to the runtime of our tool. As demonstrated in the benchmark results mentioned earlier, our tool lags behind Varscan in terms of performance. This discrepancy in runtime could become even more pronounced when handling larger samples. To address this issue, it is crucial for us to focus on further enhancing our filtering logic and optimizing the filtering order. By doing so, we can significantly improve the speed at which Vcall operates.

Another aspect that requires attention is our tool's space usage, which currently falls short compared to Varscan. To mitigate this, we should explore strategies to remove unnecessary memory allocations whenever feasible. This approach will help us eliminate redundant data storage, ultimately reducing the overall space requirements.

Furthermore, it is important to acknowledge that our understanding of how Varscan calculates statistics and filters data is limited. We developed our equations and filtering algorithms by referencing VCF manuals and Varscan manuals. However, as we deepen our knowledge in the field of statistics and gain more insights into SNP analysis, it is imperative to refine and enhance these equations and algorithms. This iterative process will allow us to improve the accuracy and effectiveness of our tool.

Finally, there are some features that is included in Varscan while we are lacking on. For example, we do not put anything for ID because it requires codes for datamining outside resources or advanced python packages to retrieved those information.

References:

[VarScan](https://varscan.sourceforge.net/)

[VCF](https://samtools.github.io/hts-specs/VCFv4.2.pdf)

[samtools](http://www.htslib.org/)

[mpileup](http://www.htslib.org/doc/samtools-mpileup.html)

[GeneCards](https://www.genecards.org)

[CSE 185 Lab 1 assignment procedures]

[Real data on *Saccharomyces cerevisiae*] (https://bioinfogp.cnb.csic.es/files/samples/rnaseq/)

https://www.proteinatlas.org/ENSG00000131591-C1orf159/pathology

Sha, Y., Wei, X., Ding, L. *et al.* Biallelic mutations of *CFAP74* may cause human primary ciliary dyskinesia and MMAF phenotype. *J Hum Genet* **65**, 961–969 (2020). https://doi.org/10.1038/s10038-020-0790-2

Biebach, L., Cindrić, S., Koenig, J., Aprea, I., Dougherty, G. W., Raidt, J., Bracht, D., Ruppel, R., Schreiber, J., Hjeij, R., Olbrich, H., & Omran, H. (2022). Recessive mutations in cfap74 cause primary ciliary dyskinesia with normal ciliary ultrastructure. American Journal of Respiratory Cell and Molecular Biology, 67(3), 409–413. https://doi.org/10.1165/rcmb.2022-0032le