A Multi-Caller Pipeline to maximize the output of Somatic Exome Sequencing Analysis

Inês Sofia Pinheiro Marques
Mestrado em Bioinformática e Biologia Computacional
Departamento de Ciência dos Computadores
2023

Supervisor
Carla Oliveira, Principal Investigator, i3S, Affiliated Professor, FMUP

Co-supervisor
Pedro Ferreira, Assistant Professor, FCUP

Acknowledgements

In first place I would like to thanks to my supervisor, Dra. Carla Oliveira, for the opportunity to work in her lab, for the knowledge, patience, help and time spent with me during the development of this thesis, and for giving me the opportunity to attend events that improved my curriculum. I would like to thank MS. Marta Ferreira, for accompanying me closely in every step of this work, for all the effort put into helping me and for the suggestions that helped me achieve the work presented here. I would also like to acknowledge my co-supervisor, Dr. Pedro Ferreira, for the insights giving in what should I focus during thesis, and also acknowledge every member of the Expression Regulation in Cancer group from i3S for the help given and the good moments.

I want to thank my parents and sister for their support.

Resumo

Os avanços na sequenciação de nova geração (NGS) melhoraram bastante a deteção de variações no DNA, quer no caso de variantes herdadas quer no caso de variantes adquiridas. O estudo destas variantes é de grande interesse na área da genética, fornecendo informações sobre o aparecimento e progressão de doenças, como o cancro, e, portanto, ajudando também a encontrar melhores soluções de diagnóstico e tratamento para estas. A identificação destas variações genómicas é feita através de múltiplas ferramentas e programas de bioinformática, um processo geralmente denominado por *variant calling*. Programas específicos desenvolvidos para esta tarefa tentam "chamar" mutações com alta confiança enquanto tentam resolver ao mesmo tempo problemas inerentes aos métodos de sequenciação, como o ruído. De acordo com o tipo de sequenciação realizada e o tipo de mutações que se pretende identificar, existem diferentes programas disponíveis para este efeito.

Com este trabalho apresentamos uma possível solução para melhorar os resultados obtidos por *variant calling* através do desenvolvimento de um pipeline composta por diferentes programas adaptados a sequenciadores específicos, além de combinar os resultados de mais do que um *variant caller* de forma a obter uma maior confiança nos resultados. Esta pipeline foi projetada especificamente para dados provenientes de exomas somáticos obtidos por meio de sequenciação do exoma (*Whole Exome Sequencing* (WES)). O nosso objetivo principal com este trabalho é tentar encontrar variantes estruturais (SV) verdadeiras com múltiplos callers em duas plataformas de sequenciação diferentes, aumentando a confiança e as informações obtidas com este método. Para atingir este objetivo, nós revimos a literatura disponível sobre os softwares usados para análise de exomas somáticos e selecionamos os programas que iam de encontro aos nossos objetivos, com posterior implementação dos mesmos. A pipeline possui diferentes funções escritas em Python para cada tarefa diferente inerente ao processo de *variant calling*, fazendo também uso de comandos Bash e R. Usamos uma amostra de um par normal-tumor sequenciada por WES nas plataformas Illumina e Ion Torrent para testar a pipeline.

Com este trabalho queremos destacar a importância do *variant calling*, bem como do uso de mais de que um *caller* para obter variantes verdadeiras com alta confiança.

Palavras-chave: Bioinformática, NGS, Somático, SV, Variant Calling, WES

Abstract

Next-generation sequencing (NGS) breakthrough improved greatly the detection of variations in the DNA, in both inherited and acquired situations. The study of these variants is of great interest in the genetic fields, giving insights on diseases appearance and progressing, like cancer, and therefore accounting for better diagnosis and treatment solutions. The identification of these genomic variations is made through multiple tools and bioinformatic programs, a process usually called variant calling. Specific programs designed for this mean try to call mutations with high confidence while tackling problems inherent to the sequencing methods, such as noise. According to the type of sequencing performed and the type of mutations desired to identify, there are different programs available for this mean.

With this work we present a possible solution to improve variant calling results by developing a pipeline that has different programs tailored to specific sequencers machines, as well as combining the results of more than one variant caller to achieve higher confidence. This pipeline was specifically designed for somatic exomes obtained through Whole Exome Sequencing (WES). Our main goal with this work is to try to find true structural variants (SV) with multiple callers in two different sequencing platforms, increasing the confidence and the information obtained with this method. To achieve this objective, we have reviewed the literature regarding software's for analysis of somatic exomes, and selected the programs that met our objectives, with further implementation of them. The pipeline has different functions written in Python for each different task inherent to the variant calling process, making also use of Bash and R commands. We used a normal-tumour pair sample sequenced by WES on Illumina and Ion Torrent platforms to test the pipeline.

With this work we want to highlight the importance of variant calling as well as the use of more than one variant caller to obtain true high confidence variants.

Keywords: Bioinformatics, NGS, Somatic, SV, Variant Calling, WES

Table of Contents

List of Tables

List of Figures

List of Abbreviations

NGS	*Next Generation Sequencing*
PCR	*Polymerase Chain Reaction*
SNV	*Single Nucleotide Variants*
SNP	*Single Nucleotide Polymorphisms*
Indel	*Insertion/deletion*
CNV	*Copy Number Variants*
WGS	*Whole Genome Sequencing*
WES	*Whole Exome Sequencing*
SBL	*Sequencing by Ligation*
SBS	*Sequencing by Synthesis*
SAM	*Sequence Alignment/Map*
BAM	*Binary Alignment/Map*
VCF	*Variant Calling Format*
TSV	*Tab-separated Values File*
GFF	*Generic Feature Format*
GTF	*Generic Transfer Format*
GATK	*Genotype Analysis Toolkit*
VEP	*Variant Effect Predictor*
CNN	*Convolutional Neural Network*
CBS	*Circular Binary Segmentation*
GUI	*Graphical user interface*
CLI	*Command-line interface*
BQSR	*Base Quality Score Recalibration*
REST	*Representational State Transfer*
API	*Application Program Interface*

1. Introduction

1.1. Exome analysis to variants detection

The genome is the complete set of genetic information present in an organism, individually arranged to provide individual characteristics. The human genome of each cell is composed of approximately three billion molecules of DNA stored in twenty-three pairs of chromosomes[1].

DNA is a molecule composed of four nucleotides (adenine (A), cytosine (C), guanine (G) and thymine (T)) and has a twisted structure in the shape of a double helix. Is composed of two different regions, the coding and the non-coding regions[1]. The coding regions, are the DNA sub-sequences of the gene that encode genes with potential to codify proteins, also called exons, whereas the non-coding regions are the sub-sequences of DNA that encode for functional RNAs but that do not have the potential to codify proteins. These latter regions also originate other DNA elements such as promoters, regulatory elements, introns, untranslated regions, centromeres and telomeres. The complete set of exons of an organism, the exome, only represents about one to two percent of the complete genome[2], which means that only this small percentage corresponds to the codifying portion of the genome.

DNA can suffer alterations in its sequence, both in the coding and non-coding regions of the genes, creating genomic variations that differ from individual to individual. These variations are important because they account for the diversity between individuals of the same species. It is believed that in humans 4.1 to 5 million bases of the genome differ from the reference genome[3]. These changes are derived from changes in the replication of DNA. The replication is based in the two strands of DNA of a mother cell, which are split and used as reference to form the complementary strand of DNA.

Variations can be classified according to where they occur. Variants that occur in gametes are called germline and are the ones passed on to the offspring[4], while variants that occur in all cells except the gametes are called somatic and are specific to the individual that has these variants, therefore not being passed to the descendants. Germline mutations are the ones responsible for heritable diseases whereas somatic mutations are known to be involved in cellular aging and many processes involved in disease progression, namely in cancer, as shown in literature[9,10]. Somatic mutations in oncogenes, tumor suppressor and DNA repairing genes increase the risk of cancer, making these types of mutations one of the most studied worldwide[11].

Variants can also be classified according to their size and type of change they produce. The smallest and most common type of variants are the Single Nucleotide Variants (SNV) (**Fig. 1 a)**), where one nucleotide of a DNA sequence is replaced by a different one. If the nucleotide that is changed is replaced by another that has the same chemical type of nucleotide base, for example, an A is changed for a G, or a T for a C, it is called Transition. But if it is replaced by a nucleotide with another chemical base type it is called Transversion, for example, an A being replaced by a C[12].

In the specific case of SNVs that occur in the coding regions of the DNA, they can additionally be classified according to the effect they have on the codon, that is, the effect they have in the nucleotide's triplets of the DNA sequence that will be translated into amino-acid sequences. Following that, if the nucleotide replacement does not affect the amino-acid sequence produced by the codon, due to the genetic code redundancy, then it is classified as a synonymous or silent variant. But if the variant leads to an alteration on the codon it can be one of two possible cases. If the altered codon produces a different amino acid, is a missense variant. If the original codon becomes a stop-coding, further resulting in a truncated-protein, it is a nonsense variant[13].

Still regarding the SNVs, is important to refer that if they are present in at least one percent of the population they are considered Single Nucleotide Polymorphisms (SNP)[4].

Within the group of small variants, but bigger than the SNVs, are the indels, insertions or deletions of less than fifty nucleotides (**Fig. 1 b)**). A specific subtype of variant caused by the indels is the frameshift variant, that is characterized by a change in the reading frame of the nucleotides that follow the indel occurrence. A reading frame is a way to divide the genetic code into non-overlapping sets of codons that will form the coding region of the gene, and can start in one of three possible positions, which means, in one of the three nucleotides present in the start codon of the coding region. This means that if the length of the indel is not a multiple of three, these reading frames will shift the reading starting positions, originating these types of variants and leading to incorrect or incomplete translations of the gene in cause[14].

Lastly, the structural variants, are the largest variants, affecting more than fifty nucleotides (**Fig. 1 c)**). These types of variants include the copy number variants (CNVs), where the total number of nucleotides is altered due to duplication or deletion events, changing the normal number of copies of specific regions in the genome. Likewise, insertions, inversions and translocations of more than fifty base pairs cause changes in the structure of chromosomes[4,5].

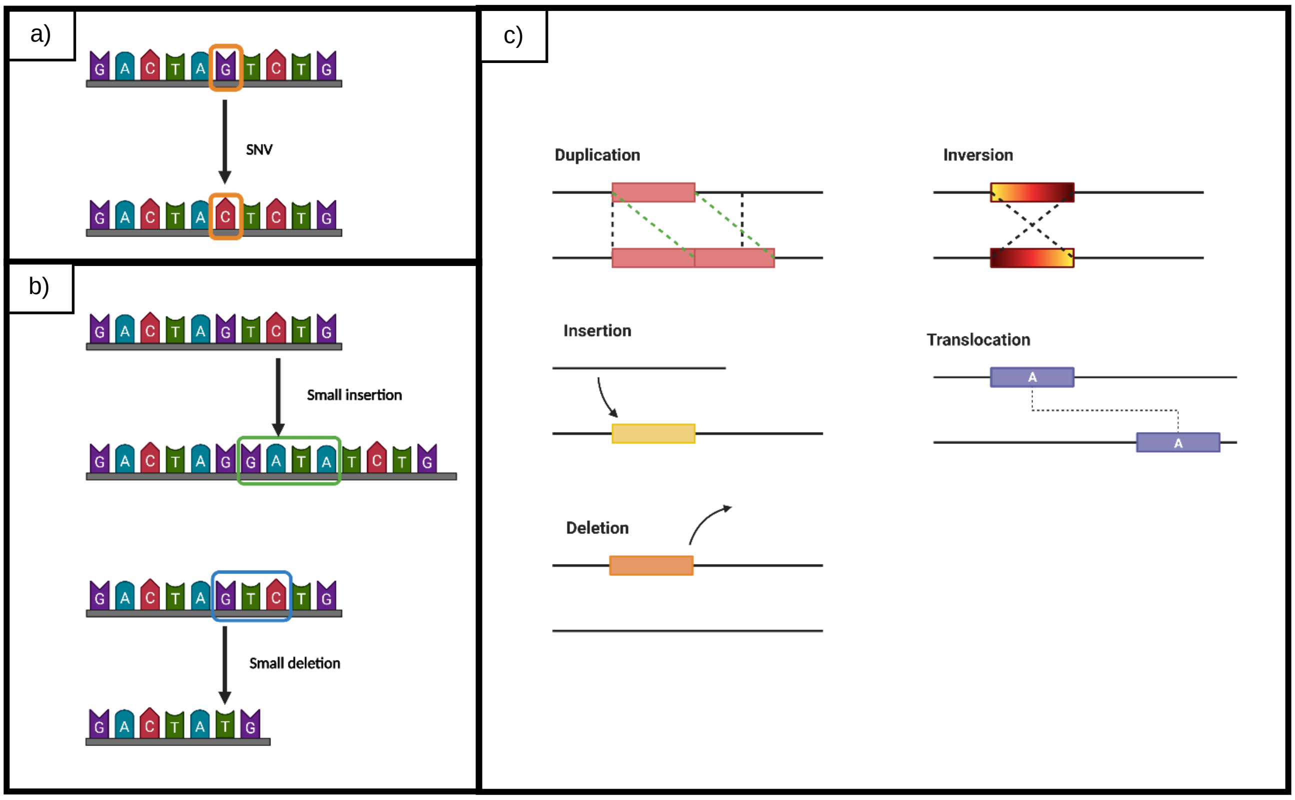


Figure 1 - Types of variants. a) SNV, b) Indels and c) SV

Of all the variants present in the human genome, 99.9% are SNPs and indels; however structural variants affect more bases, around 20 million, due to their characteristics[5].

In some cases, variations can confer advantages to the individuals that carry them, therefore being benign, these are shared in the further generations and become common in the population, normally called polymorphisms. However, they can also be pathogenic, these ones have impact in the normal function of the cells that can lead to disease. It is estimated that throughout the life of an individual, eighty-five percent of his exome suffers DNA mutations[2].

1.2. Whole Exome Sequencing (WES)

Next-generation Sequencing (NGS), or Massive Parallel Sequencing, is a high-throughput DNA and RNA sequencing technology that revolutionized the genetic analysis field and that is commonly used worldwide, namely in a clinical context. Sequencing technologies are used to identify the order of the nucleotides of regions or complete sequences of DNA and RNA. The major characteristic of NGS, and as its name suggests, is that it allows the sequencing in parallel of more than one sample simultaneously, something that the previous sequencing methods, such as the Sanger sequencing, did not allow. Still in comparison with the Sanger sequencing, NGS is more cost-effective and quicker, and achieves better accuracy and confidence in the results. Furthermore, it offers single-nucleotide resolution, allowing the detection of SNVs and larger variations, as well as the detection of mosaicism and low-frequency variants[15].

A common NGS workflow for DNA is composed of four steps. The first step is the library preparation where the DNA molecules of each sample are randomly fragmented into shorter fragments, either enzymatically or by sonication (applying ultrasound waves to break the chemical bonds present on the DNA). Adaptors are then ligated to these fragments by the DNA Ligase, forming a library for each sample. A common practice inside this step is “pooling” or “multiplexing”, that is, to mark the adaptors with a specific DNA barcode, so each sample can be easily tracked throughout the process. The next step is clonal amplification. In this step the DNA molecules from the previously prepared libraries are attached to a solid surface, such as beads or flow-cells, and amplified by PCR, creating identical clones of each molecule so their signal is increased and they are easily detected during the following step, sequencing. This step varies from platform to platform. The first NGS sequencing approach used was developed by Roche/454, and used a pyrosequencing method, a pioneer of the Sequencing by synthesis method. In this method pyrophosphate molecules would be released when a nucleotide was incorporated in the new DNA strand. Nowadays this method is no longer used, with the current ones being the following. Sequencing by synthesis, used by Illumina, in which nucleotides with reversible fluorescent blockers bind to the DNA template strand, releasing a fluorescence signal upon incorporation that is specific to each nucleotide base. Pictures of the chip are then taken after every synthesis round and a computer identifies what bases were added according to the color that appears on the picture. Sequencing by ligation, used by SOLiD, where the mismatch sensitivity of the DNA ligase is used to identify the nucleotide sequence in a fragment. Four fluorescent labeled probes compete to ligate to the sequencing primer, and upon ligation the fluorescence of the ligated probe is captured and identified. And lastly, Ion Torrent uses an Ion-Semiconductor Sequencing approach in which a hydrogen ion is formed upon the incorporation of the nucleotide in the new synthesized strand. These hydrogen ions will lead to a change in the pH of the environment, creating a high positive voltage that is then detected by semiconductor chips[15,16,17,19].

Whatever the sequencing technology used the determined sequences, commonly called raw reads, are saved in a file. This file is the base of the final NGS step, the analysis of the sequencing data using bioinformatic techniques. These will be explored in the next chapter.

DNA sequencing can be subdivided into three main types according to what regions of the DNA they sequence: Whole Genome Sequencing (WGS), Whole Exome Sequencing (WES), and Target Sequencing. Their main differences are stated in **table 1**.

Table 1 - Types of DNA sequencing[20,22,23]

	Whole genome	**Whole exome**	**Target sequencing**
Sequencing regions	Whole genome (codifying and non-codifying regions)	Whole exome	Specific group of genes (usually between 10 to 500)
Sequencing Coverage	> 30x	> 100x	> 500x
Cost per sample	< $1000	~$800	$250 to $500

Sequencing coverage is defined as the number of reads that align to regions of the reference genome. Higher coverages are more sensitive and therefore allow the detection of low-frequency alleles variations[24].

As mentioned in section 1.1, most disease-causing variants are found in coding regions of DNA, the exome. This makes the use of WES more suitable to study these types of variants when compared with WGS, since it has higher sequencing-depth, lower costs and requires less storage, besides only sequencing the regions of interest. On the other hand, if we are interested in variations of non-coding regions that could potentially lead to genetic disorders, then WGS is the most used method[16,18].

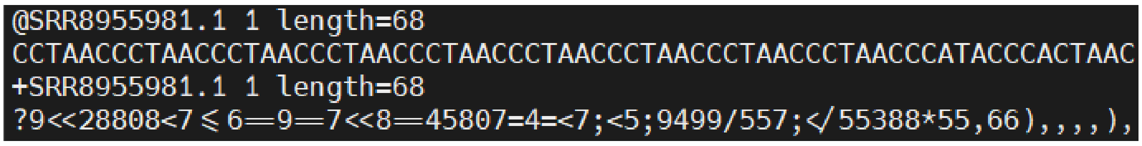
1.3. Bioinformatics and its importance in research

The fast and diverse growing of sequencing technologies caused a huge increase in available data and resources to analyze this data, this leads with emerging of a new scientific discipline, the bioinformatics. It is a discipline that develops and uses computational tools to collect, store and analyze biological data. It is a multidisciplinary field that involves knowledge from Biology, Computer Science, Chemistry, Physics and Statistics.

WES analyses are made using pipelines that usually have the following backbone: quality control, reads preprocessing, alignment, post-alignment processing, variant calling, and variant annotation[25]. A detailed explanation of each step is given below.

1.3.1. Quality Control

First of all, the file that comes out of the sequencer is analyzed regarding its quality. The output of a sequencer is usually a FASTQ file, a text file that contains sample reads and their respective Phred quality scores represented in ASCII characters, standard encoding characters used in electronics (**Fig. 2**).



```
@SRR8955981.1 1 length=68
CCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCATACCCACTAAC
+SRR8955981.1 1 length=68
?9<<28808<7⩽6═9═7<<8═45807=4=<7;<5;9499/557;</55388*55,66),,,,),
```

Figure 2 - Exemple of a FASTQ file

Even under the same sequencing conditions, files that come from sequencers can be different from each other due to chemical and instrumental failures, and therefore, it is very important to account for these errors by verifying the quality of these files and then preprocess them accordingly. Checking the quality per base, the percentage of GC content, the presence of adapters and the number of duplicated reads, are some of the most verified characteristics on the FASTQ files.

FastQC[28] is the most used program worldwide to check these files. This program accepts inputs in varied formats and from almost all of the available sequencing platforms. It outputs a graphical report in the form of HTML file, a text file specifically designed to be seen in a digital context, containing interactive contents within it. This report contains information about the number of reads, per base sequence content, per base sequence quality, per base content and nitrogen content, per sequence quality scores and GC content, sequence length distribution, sequence duplication levels, overrepresented sequences and k-mer content.
FastQC can be used in one of two modes, stand-alone interactive mode or non-interactive mode. The first has a graphical-user interface (GUI) and only outputs the HTML report whereas the second one is only through command-line interface (CLI) and

outputs a .zip folder that contains the graphs of the HMTL file and other data files in addition to the HTML report.

1.3.2. Reads preprocessing

As said previously, after the quality control of the reads they are preprocessed if needed. In this step, low-quality reads, adaptors and/or unwanted sequences are removed in order to improve the number of correctly aligned reads in the next step, the alignment[29].

In some cases, it happens that after reads processing, some of the reads on the file are longer than others. This is usually due to low-quality bases or adaptor trimming, a situation that could lead to misalignment between the sequence and the reference genome, so depending on research objective sometimes we need to choose only the reads with certain size. Contaminations can be observed through the GC content. Normally the GC content follows a normal distribution, but if this is not the case, then it can be indicative of contamination.

Trimmomatic[30] is a program that performs different trimming methods in Illumina single and paired-end data. Users can apply more than one trimming method and choose the order by which the methods will be applied to the data through the order they write the methods on the command. The methods available are the following: *ILLUMINACLIP*, removes Illumina sequencer adapters and specific sequences, with the program containing some of the most common used adapters in the *Trimmomatic0.39/adapters/* folder. Users can specify the number of mismatches the program allows per full match, the accuracy of the match between the adapter reads from the sample and the one from the file given, and the accuracy of the match between the adapter sequence and the sample sequence. *SLIDDINGWINDOW* removes reads whose quality makes the average quality of the defined search window be below a defined threshold. *LEADING* and *TRAILING* both remove reads below a threshold quality, on the start and on the end of the read respectively. *CROP* trims the reads to a specified length, and *HEADCROP* removes the specified number of bases from the start of the read. *MINLEN*, removes the read if its quality is below a threshold, and lastly, *TOPHRED33* and *TOPHRED64* convert the quality score values present on the reads to Phred-33 or Phred-64 scores respectively.

SolexaQA[31] is another trimming program composed of three different algorithms: *Analysis*, *DynamicTrim* and *LengthSort*. *Analysis* gives information about the quality and characteristics of the FASTQ files, *DynamicTrim* trims the reads and keeps their longest

contiguous segments whose quality is above a defined threshold, or uses the *BWA*[32,33] trimming algorithm, another trimming program algorithm, to perform this step. Users can specify the quality threshold both in probability values or Phred quality scores, or use the default value, that is a probability of 0.05.
The *Lengthsort* algorithm separates low-quality from high-quality reads, storing both in two different respective files, and can remove unpaired reads in paired-end mode using the -c option.
The first two algorithms have options that allow the user to specify the sequencer that originated the files, since it accepts data from Illumina, Ion Torrent or Roche/454, even though the program automatically identifies the origin of the data.
Usually, the last two algorithms are used together to preprocess reads, so we followed this recommendation.

The output of this programs is a FASTQ file.

1.3.3. Alignment

The alignment consists in aligning the reads to a reference genome or submitted to *de novo* alignment (if a reference genome does not exist or if the downstream analysis benefits from that), which consists of assembling a genome from scratch without reference. In the case of the human species, there is available a reference genome[34].

The first step of alignment is the index of the reference sequence to speed up the overall process, with each index being specific to each reference sequence and aligner. This step outputs a BAM file, a tab-delimited text file with a header, with information about the metadata (HQ), reference sequence (SQ), read group (RG) and the program used to perform the alignment (PG); and the aligned reads, that contain information about the name, sequence, and quality of the reads, as well as information about the alignment[35].

BWA[32,33], or *Burrow-Wheels Alignment Tool*, is the most used program used to align low-divergent sequences to reference genomes. The index is made through *bwa index*. With the use of this command, users can specify the algorithm used to construct the index using the argument -a, with the available options being *IS* and *bwtsw*. *IS* is the default algorithm and only works for genomes up to 2GB of size, whereas the *bwtsw* works for larger genomes, such as the human genome[32].

Regarding the alignment algorithms there are three possible choices from which the user can choose, the *bwa backtrack*, specifically designed for reads up to 100 base pairs and sequenced by Illumina platforms, the *bwa mem* and the *bwa-sw*. These last two are similar, working with reads longer than 100 bp and allowing the performance of split alignments, that are characterized by the alignment of different parts of the sequences to disjoint regions of the reference. However, *bwa mem* is more used than the *bwa-sw* for being faster and more accurate in high-quality reads when compared to the latter[32,33].

Users can specify a variety of arguments when running the alignment command such as the number of threads the program can use, the header of the output file, the minimum seed length, the matching score, the mismatch penalty, and many other options regarding the alignment scores.

Another program used for alignment is *TMAP*, a program developed by ThermoFisher specifically for the alignment of Ion Torrent data to a reference genome[36]. Just like *BWA*, it also has an option to index the reference genome as well, and it also contains more than one alignment algorithm, *map1*, *map2*, *map3*, *map4* and *mapvsw*. *map1*, *map2* and *map4* are all based on *BWA*, with the first being better for short reads and the second for long reads, and *map4* being the default algorithm for being more general. *map3* is based on SSAHA (Sequence Search and Alignment by Hashing Algorithm) and k-mers. Lastly, *mapvsw* is an implementation of the Smith-Waterman algorithm[37]. It is also possible to combine more than one algorithm using the *mapall* command, fastening the process and increasing sensitivity. Inside each algorithm there are a lot of options the user can specify such as matches score, mismatch penalties, gaps penalties, among many others.

Alignment programs take as input a reference genome FASTA file and the sample FASTQ, and output a BAM file.

1.3.4. BAMs processing

According to *GATK* Best practices[38], after the alignment it is recommended to process the file obtained in this step because during the preparation of the samples for sequencing and during the sequencing itself, duplicated reads can arise, leading to errors during variant calling. Due to this, they must be marked or removed. Furthermore, sequencers are also subject to technical errors that may over or underestimate the quality score of some sequence bases. Given that variant calling algorithms highly depend on these quality scores, it is necessary to recalibrate these latter in case there was some technical error[15]. To do that, machine learning algorithms based on empirical error models are applied to the scores. These steps are illustrated in **figure 3.**

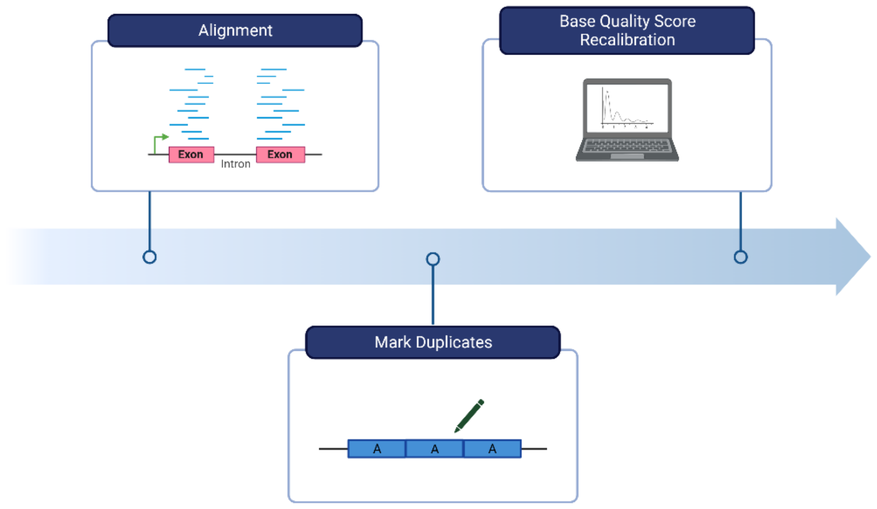


Figure 3 - BAMs processing steps

GATK is a software composed of a set of computational tools used for many tasks in the processing and discovery of variants, both germline or somatic, in DNA and RNA-seq. The use of *MarkDuplicates*, *BaseRecalibrator* and *ApplyBQSR* tools present in *GATK* is commonly used to perform the processing of BAMs. *MarkDuplicates* is used to mark the duplicates present on the reads. This tool takes as input the alignment BAM file, and outputs a BAM file with the duplicate reads marked with the tag DT, that stands for duplicate type, on the optional field of the BAM file, as well as a text file with the metrics used to consider a read a duplicate. Users can add other optional arguments to the command, such as marking all duplicates, marking only the optical ones, as well as specifying the program to remove the marked duplicates[38].

BaseRecalibrator then takes the previously obtained BAM file as input as well as the reference genome and a file with the known polymorphic sites present in the human genome, and outputs a recalibration table that will be used by *ApplyBQSR*. This last will also take as input the same BAM file and reference genomes used by *BaseRecalibrator*, and will output a BAM file with the recalibrated base scores of the reads[38].

1.3.4. Variant calling

Variant calling comprises the identification of differences between the reads and the reference genome and see where they differ. Variant callers can work on each sample individually, what facilitates the automation of this process; or they can call multiple samples simultaneously, which allows the caller to produce genotypes for all the variants of the cohort in all samples. Following this, variant calling in multiple samples simultaneously, also called joint analysis, has higher sensitivity to detect variants present in low-coverage regions[25].

The output of this step is a Variant Call Format (VCF) file, a text-file that stores the found variants[39].

The variants detected are dependent on the type of caller that is chosen, with some callers only detecting SNVs and small indels, while others only detect CNVs and structural variants. The programs are also specific to the provenance of the samples, this is, if they are from germline or somatic cells[25].

1.3.4.1. SNV/Indel variant callers

SNV/Indel variant callers use one of two methods, heuristic or probabilistic methods[40]. Heuristic methods were the ones used in the early days of variant callers, when these would make their calls according to fixed cut-offs of heuristic factors like base quality, alignment quality and coverage. This usually leads to miss variants from heterozygous genotypes when the read depth is low or medium. Apart from that, callers that use these methods do not produce any type of confidence measure regarding about the called genotypes. On the other hand, probabilistic methods make use of probabilistic functions to calculate the likelihoods of the possible genotypes, followed by the score of a certain genotype being expected in a given place. One of the most used functions is based on the Bayesian inference method. In this, the genotype's likelihoods are obtained to then calculate the prior probability **P(G|R)** of a genotype **G** happening given the information of the read count coverage **R**.

$$P(G|R) = \frac{P(R|G)P(G)}{P(R)}$$

Where **P(R|G)** is the previous calculated genotype likelihood and **P(G)** is the prior probability of that genotype happening, and can be obtained from external databases, the reference genome or can be assumed as the same for all genotypes. The result probability is used either directly or indirectly as a confidence measure. The genotype with the highest value of prior probability is chosen as the possible genotype at that site[40].

Machine learning algorithms, namely deep learning ones, that are algorithms based on artificial neural networks (ANNs) are recently being used to perform variant calling[41].

In **table 2** we present some of the available SNV/Indel variant callers.

Table 2 - SNV/Indel variant callers

Name	Operative System	Sequencing Type	Algorithm	Number of Citations	Year
DeepVariant[41]	Linux, MacOS	WGS, WES	CNN (Convolutional Neural Network)	736	2017
FreeBayes[46]	Linux	WGS, WES	Bayesian Model	-	2012
GATK HaplotypeCaller[47]	Linux, MacOS, Windows	WGS, WES	Bayesian Model	1111	2015
SomaticSniper[48]	Linux, MacOS,	WGS, WES	MAQ Genotype Likelihood Model	510	2011
Strelka2[49]	Linux	WGS, WES	-	796	2018
VarScan2[50]	Linux, MacOS	WGS, WES	Heuristic method plus Fisher's Exact Test	3765	2012

- **DeepVariant**

DeepVariant is a variant caller based on Convolutional Neural Networks (CNN), a deep-learning method. It takes the input files, in either BAM or CRAM format, and finds possible variants, that are then transformed into tensors, that are matrixes with multiple dimensions that represent an image. CNN is applied to the tensors, assigning a genotypic likelihood for three possible genotypes, homozygous reference, homozygous alternate or heterozygous alternate[41].

- **FreeBayes**

FreeBayes is a haplotype-based algorithm that uses BAM input files with short-read alignments from the individuals of a population, and a reference genome, and determines the most likely genotype in the population in a given position[46]. **Figure 4** exemplifies this method.

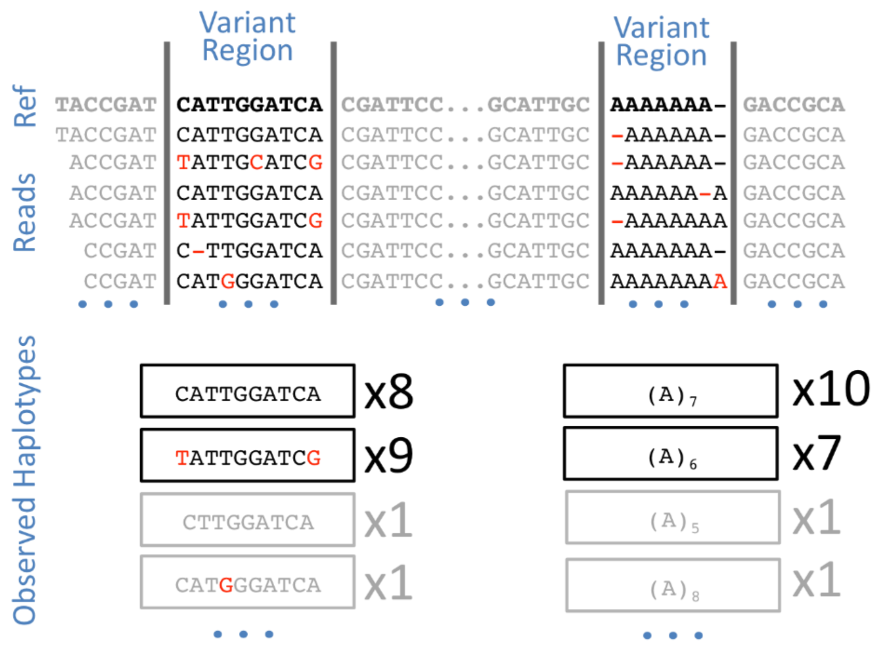

Variant Region
Variant Region
Ref
Reads
Observed Haplotypes
TACCGAT CATTGGATCA CGATTCC...GCATTGC AAAAAAA- GACCGCA
TACCGAT CATTGGATCA CGATTCC...GCATTGC -AAAAAA- GACCGCA
ACCGAT TATTGCATCG CGATTCC...GCATTGC -AAAAAA- GACCGCA
ACCGAT CATTGGATCA CGATTCC...GCATTGC AAAAAA-A GACCGCA
ACCGAT TATTGGATCG CGATTCC...GCATTGC -AAAAAAA GACCGCA
CCGAT C-TTGGATCA CGATTCC...GCATTGC AAAAAAA- GACCGCA
CCGAT CATGGGATCA CGATTCC...GCATTGC AAAAAAAA GACCGCA
CATTGGATCA x8
TATTGGATCG x9
CTTGGATCA x1
CATGGGATCA x1
(A)7 x10
(A)6 x7
(A)5 x1
(A)8 x1

Figure 4 - FreeBayes method. Adapted [46]

1.3.4.2. SV variant callers

SV variant callers are based in four main methods: read-pair, read-depth, split-read and local-assembly (**Fig. 5**). Read-pair is based on the insert size present on the sequenced read-pairs and the observed on the reference genome. Split-reads is based on the complete aligned of a read and a partial or fail alignment of the other read. Read depth is based on the correlation of the read depth of a region with the number of copies of that region. And lastly, local assembly, that generates contigs, without looking to the reference, and then comparing this contigs with a reference[51].

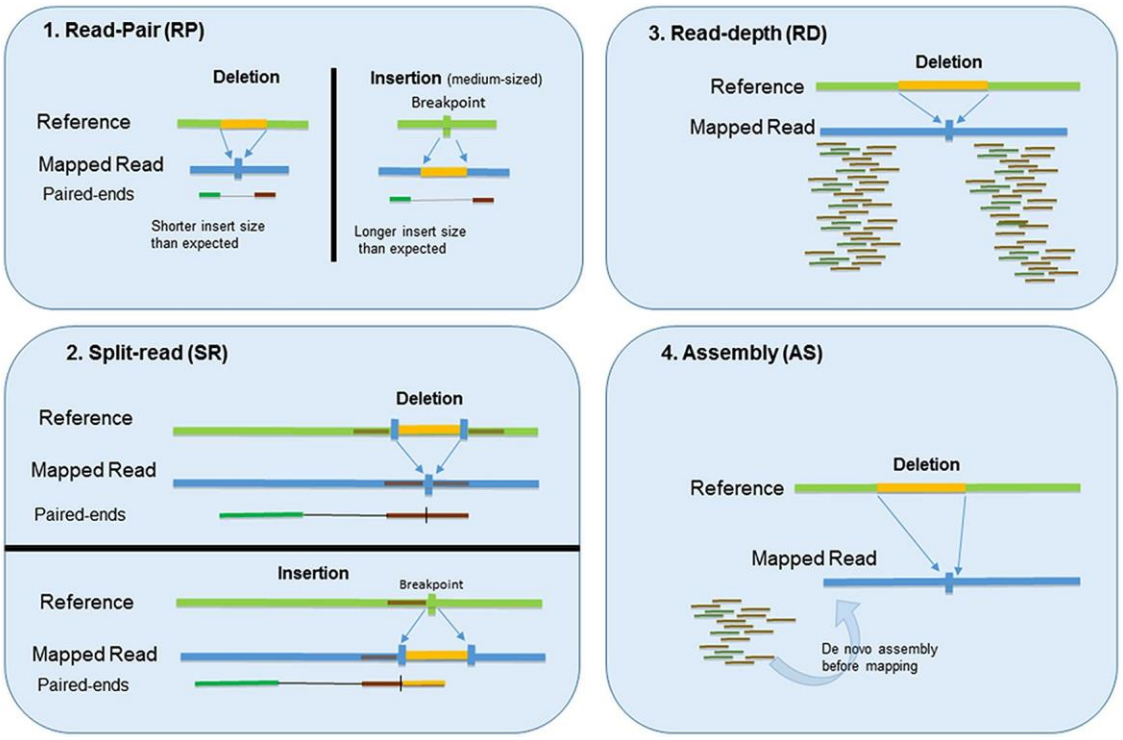


Figure 5 - Methods used for SV detection[52].

A summary of some of the most used SV variant callers is present on **table 3**.

Table 3 - SV variant callers

Name	Operative System	Sequencing Type	Sequencer	Algorithm	Number of Citations	Year
cn.MOPS[53]	Linux, MacOS, Windows	WGS, WES	Illumina, Roche	Bayesian Model	366	2012
CONTRA[58]	Linux, MacOS,	WGS, WES	Illumina, Roche	Bayesian Model	283	2012
DELLY[59]	Linux, MacOS	WGS	Illumina, PacBio	Split and paired reads based	1582	2012
ExomeDepth[60]	Linux, MacOS, Windows	WES	Illumina	Read Depth	493	2012
VarScan2[50]	Linux, MacOS	WGS, WES	Illumina, Ion Torrent	Heuristic method plus Fisher's Exact Test	3765	2012

- **Contra**

Contra (Copy Number Targeted Resequencing Analysis) identifies copy number variants, including large variants. It detects copy number gains and losses by estimating log-ratios variations through binning and interpolation methods. It requires as input a BAM or SAM alignment file for both control and tumor samples, a BED file with the targeted regions where the sequenced regions are reported, and the reference genome. Users can specify optional parameters such as the minimum read depth, minimum number of bases and p-value threshold, among others, as well as run the CBS (Circular Binary Segmentation) algorithm and create a plot of log-ratio distributions for each bin[58].

- **VarScan2**

VarScan2 works in all major sequencing platforms and can identify SNVs, indels, CNVs and LOHs in both germline and somatic samples. To identify the variants, the program uses a heuristic/statistic approach that identifies which variants meet the various thresholds of the base quality, read depth, statistical significance and variant allele frequency parameters. It takes an alignment file from both the normal and tumor sample, and the reference genome.

VarScan has a recommended workflow. The first step is to run the *copynumber* algorithm on both normal and tumor alignments. Then ran the *copycaller* algorithm that makes preliminary variant callings and adjusts the GC content, followed by the application of a CBS algorithm from the R BioConductor library DNACopy. Lastly, adjacent segments with similar copy number values are merged and then classified according to their size. The *copycaller* algorithm can accept additional parameters such as the minimum coverage and minimum region size threshold, the lower and upper bound for log-ratios to be considered amplifications or deletions, respectively, and, in case it's necessary to recenter the alignment data around a baseline.[50]

- **GATK**

Another possible way to detect copy ratio alterations is using *GATK*. As said previously, *GATK* encompasses several tools. With the combination of some of these tools it is possible to detect copy ratios alterations. To achieve that, a "Sensitively detect copy ratio alterations and allelic segments" tutorial is available on the *GATK* website[61].
The first step of this tutorial is to collect coverage counts from the alignment files. To do that a binning process is applied in a file with genomic regions intervals, using the *PreProcessIntervals* tool, and then the coverage counts of the pair-end fragments that fall within the bins present in the previously prepared file are collected, using the *CollectReadCounts* tool. The following step is to create a panel of normals (PON). A panel of normal makes use of the normal samples to define the baseline level from which the CNVs are called. To achieve that the *CreateReadCountPanelOfNormals* tool is used, that takes as input the normal samples counts obtained previously and outputs a PoN in HDF5 format, a data type format that allows the storage of more than one type of data in a single file. With the obtained PoN, read counts are denoised in order to obtain denoised copy ratios by using the *DenoiseReadCounts* tool. This tool can perform additional GC-bias correction in the copy ratios.

The next step is to count the reference and alternative alleles at common germline variant sites. This is done using *CollectAllelicCounts*, for both the normal and tumour samples. Copy and allelic ratios that are contiguous are then grouped using

ModelSegments. To do this the tool performs three steps. First it identifies the heterozygous sites and removes the ones that overlap with copy ratio intervals. Then performs a multidimensional kernel segmentation followed by a Markov-Chain Monte Carlo sampling and segmentation smoothing. Lastly, CNVs are called using the *CallCopyRatioSegments* tool[61].

1.3.5 Variant Annotation

Variant annotation is an important step that allows the collection of functional information associated with each of the identified variants, as well as add information regarding the genes, transcripts, proteins, and known variants. Moreover, this information together with information about allelic frequencies on specific population, association to diseases, among others, help on the inference of the effect these variants can have at various biological levels. Since this step is very important for identifying meaningful variants, the databases that contain additional information should be chosen very carefully[62]. There are a lot of databases available to predict the variants effect or add additional information to the reported variants, such as CADD[63] (Combined Annotation Dependent Depletion), DGV[64] (Database of Genomic Variants), GENCODE[65], gnomAD[66] (The Genome Aggregation Database), OMIM[67] (Online Mendelian Inheritance in Man), among others. This databases contain information about many of the previous stated genomic features of our genome, that are then used by the annotation tools to compare that information with the variants obtained through variant calling and identify differences and similarities, therefore assigning genomic information to the variants identified. Most of the available tools to perform variant annotation make use of one or more of this databases to perform this step, being conditioned by the information present on these databases.

- **AnnotSV[68]**

AnnotSV is an annotation tool that compiles information about structural variants function, regulation and clinical relevancy. It starts by identifying if there is overlap between any variant identified in the previous step and the transcripts present in the human genome build GRh37 or GRCh38. Then if the variants are associated with a gene, their names are added to the annotation file, and lastly, two annotations are generated for each SV identified, one that is based on the complete SV, and other that is based on the genes present within the SV. *AnnotSV* can perform three types of annotations: genomic, patient or custom-based. Genomic-based annotation makes use of databases such as DGV[64] and RefSeq[69] to find genes or transcripts that overlap the SVs, and ClinVar[70], ClinGen[71] and OMIM[67] databases to report overlaps with known pathogenic genes or genomic regions. Patient-based annotations make use of the SNV and indels variants identified in the patient's data to annotate true SVs and discard false positives, and custom-based use a user-given tab-separated file with the information the user wants to annotate. If identified variants overlap with known pathogenic genes or genomic regions when using genomic-based annotations, *AnnotSV* will rank the variant in a scale of five classes that go from "Benign" to "Pathogenic", according to recommendations from the America College of Medical Genetics (ACMG) and ClinGen[71,72]. The output of this tool is a tab-separated values file (tsv), and it can also output a VCF file optionally.

- **Ensembl Variant Effect Predictor (VEP)[73]**

VEP gives information about the effect of SNP/indels and SV in genes, transcripts, protein sequences and regulatory regions. Users can choose which database to use from a variety of databases present within this tool, or can use GFF (Generic Feature Format) or GTF (General Transfer Format) files with transcripts information present on them. Users can also connect to MySQL database servers that contain Ensembl databases.
VEP allows the use of more databases that are not present in the program cache by installing them through "plug-ins".
If the variants obtained in variant calling are known or overlap known variants, *VEP* also gives information regarding allele frequencies, phenotypes and associated diseases. Information regarding the allele frequencies is obtained from the 1000 Genomes Project[3], and the SIFT[74] and PolyPhen2[75] tools score the variants identified and use this score to predict the effect of the variant in amino acid sequences, and possible amino acid substitution.
The output of this tool is a VCF, a TXT or the tool specific format, VEP format[73].

1.3.6. Variant Prioritization

Variant callers retrieve millions of variants, but that does not necessarily mean that all of them must be trusted. Filters are usually necessary to obtain high-confidence variants, however, most of these filters rely on metrics that are highly variable according to the caller used, such as genotyping quality metrics. Some programs already apply filters, and add a flag if the variant passes those filters. However, this is not linear, and the high-confidence variants obtained must always be verified by biologists and clinicians. On the other hand, using population frequency, genomic location and variant class is normally used to filter variants, since a rare variant is more likely to have more impact than a frequent[74].

1.3.7. Variant Visualization

Lastly, variants are visualized. This step allows researchers to obtain better insights about the data[77]. The tools available for this means allow the visualization of the aligned reads, with the most common ones being the linear genome browsers such as *IGV* (Integrative Genome Viewer)[78] or *UCSC Genome Browser*[79] (**Fig. 6 A)**). These display the reference genome and user's reads in genomic intervals that allow for zooms in them, enabling the visualization of changes at the single-nucleotide level[80].

Circo plots and linear coordinate plots are also of great interest to visualize variants (**Fig. 6 E) and F)**). The first one represents the 22 chromosomes, as well as the X and Y ones, as arcs of a circle, with the number of SVs in each chromosome being represented as curves. Linear coordinate plots represent one or more chromosomes, with lines or curves connecting and representing the two end-points of the SVs[80].

A simple yet effective way of visualizing the variants is also through a table (**Fig 6 C)**).

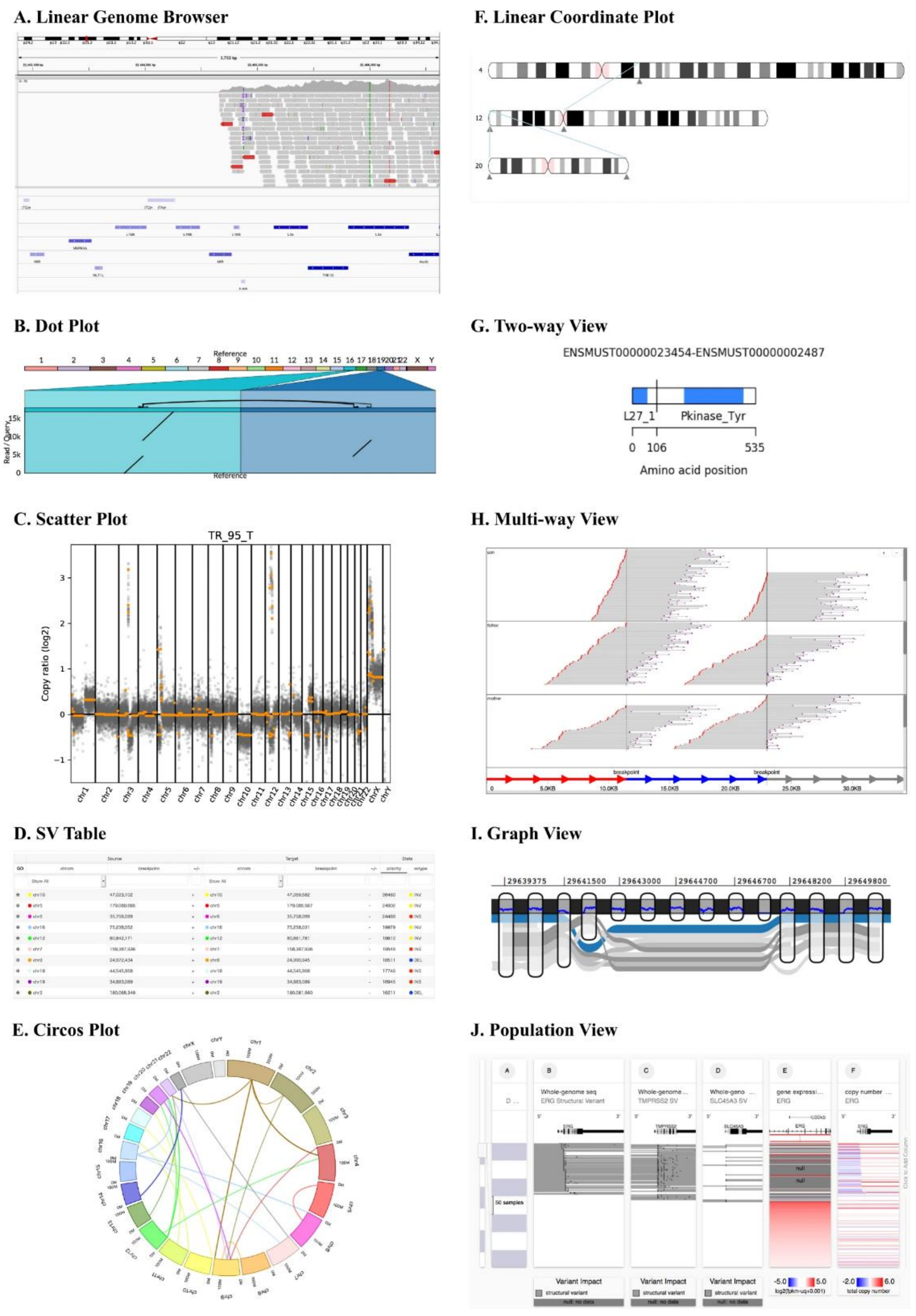


Figure 6 - Variant visualization methods. Adapted[80]

2. Aim

Variant calling is an important method to obtain insights regarding unknown variants and unravel potential effects that come from their existence. Moreover, the knowledge gained from analyzing the data that comes from this process has been proven to help with the development of treatments and clinical diagnostics for some diseases. Nonetheless, NGS methods, either WGS or WES, are prone to errors, and the results obtained from variant calling of these types of data must be verified carefully and validated through Sanger Sequencing, a method that is costly and time-consuming, and that is not error-free as well[81].

Due to this, it is important that the results obtained from variant calling are of the highest-confidence possible so the insights that can come from them are accurate. As stated in the literature, one way to achieve this is by using more than one variant caller and combine their results to obtain a high confidence set of true variants[25].

Another factor that can influence the variants obtained through this process is the sequencer used to obtain the NGS data. Different sequencers use different methods to prepare the libraries and different technologies for the process of sequencing itself which can lead to different variants called.

Therefore, the main goal of this work is to use multiple callers in the same data that comes from two different sequencers, to find a set of high-confidence true variants.

3. Materials and Methods

3.1. Literature review

Our work started with a review of the literature available regarding somatic variant callers. In a first stage we collected the available literature and analyzed it for possible programs to integrate our pipeline. Potential programs were obtained by considering the possibility of accepting data from Illumina and/or Ion Torrent, and being available to download at the time of development of this work. We then analyzed the literature for each program individual and compared them relatively to metrics such as computational time they take, F1-score, number of citations and if they were already benchmarked.

3.2. Pipeline development

We developed a bioinformatic pipeline on data from two different sequencers, Illumina and Ion Torrent, two of the most worldwide used sequencing companies. Our pipeline was written in Python but incorporates Bash and R commands through the use of Python packages. The workflow of the pipeline was developed so it could adapt to the type of sequencer used to obtain the initial data, making use of different programs in some of its steps, due to their better performance on data from one sequencer when compared to the other. The workflow of our pipeline is shown in **figure 7.**

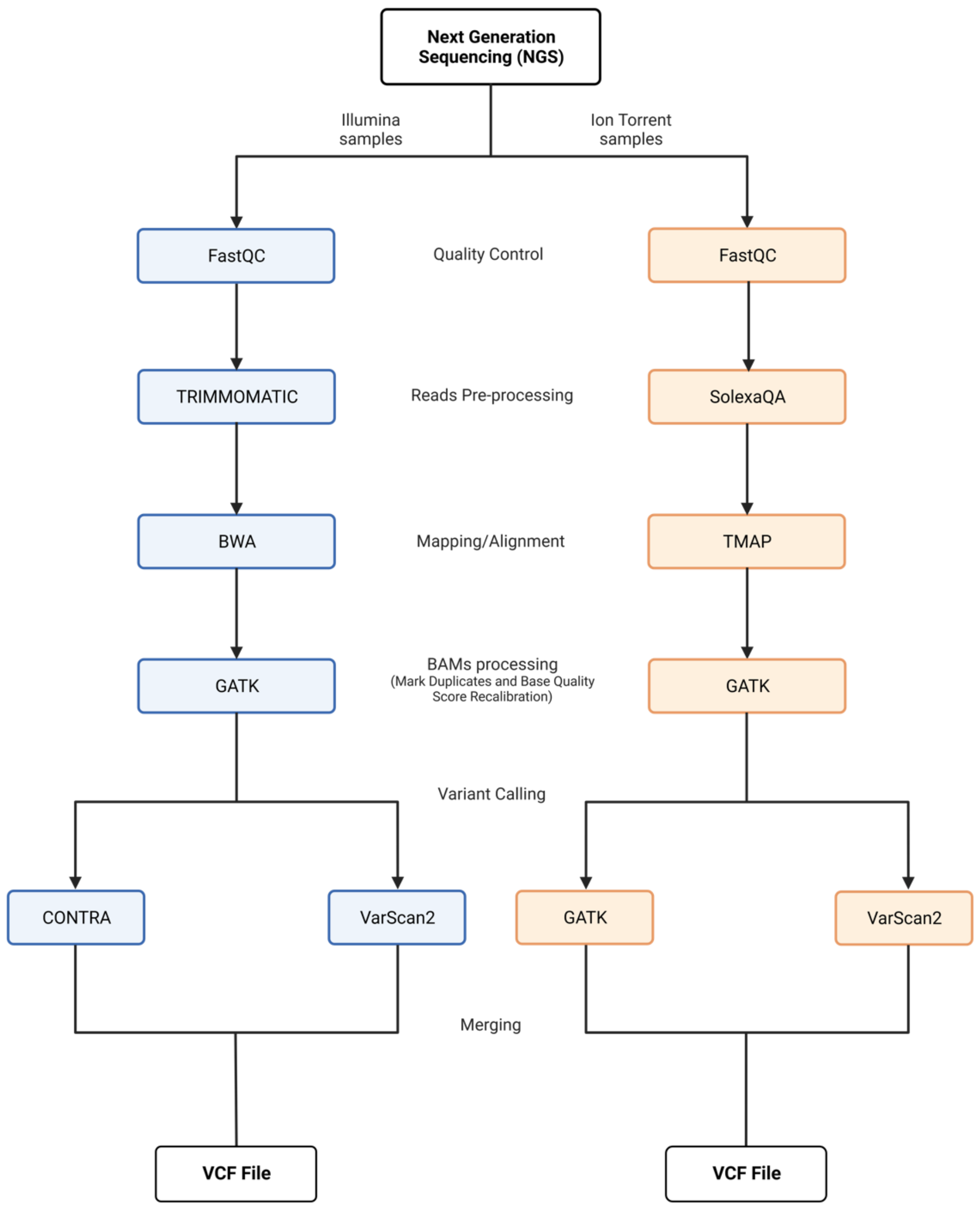


Figure 7 - Pipeline workflow

3.2.1. Samples and Reference files

To develop the pipeline, we used data from the Sequencing Quality Control 2 (SEQC2) project[82] of the Microarray and Sequencing Quality Control (MAQC) consortium[83]. We choose two sequencing replicates of a normal and tumor paired samples sequenced by WES. The normal sample (https://www.ncbi.nlm.nih.gov/biosample/SAMN10102574) was a B lymphoblast cell line, and the tumor sample (https://www.ncbi.nlm.nih.gov/biosample/SAMN10102573) a triple-negative Breast Carcinoma cell line, both from the same donor, a 43-year-old woman. The samples replicates were sequenced both on an Illumina and Ion Torrent Sequencer. The samples obtained by the Illumina Hiseq 2500 platform had the SRA accession number SRR7890845 (normal) and SRR7890844 (tumor), and the samples obtained by the Ion Torrent Ion S5 platform had the SRA number SRR8955981 (normal) and SRR8955982 (tumor). Illumina replicates were paired end meaning there were two reads per sample. Their FASTQ files were downloaded from the NCBI website (https://www.ncbi.nlm.nih.gov/sra) using the SRA Toolkit .

Besides the samples files we also needed a reference genome. We used the hg38 version of the Human Genome obtained from the Genomic Data Commons website.

Some steps also required a file with the known-polymorphic sites of the human genome, which we used the last dbSNP build, dbSNP 150, obtained from the NIH Medical Genetics and Human Variation website. A file with the target regions used in the variant calling step is also needed, which was obtained from the available data of the article where the samples we used were described.

3.2.2. Quality Control

We inspected the quality of all FASTQ files, for both sequencers' replicates, with *FastQC*[28], using the command-line version since is the most suitable for pipelines. As said previously, this tools reports information about per-base sequence content and quality, duplication levels, and overrepresented sequences, among others. We checked the results obtained on the reports and proceeded to pre-process the files according to what was observed.

3.2.3. Reads Processing

Files obtained from the Illumina sequencer were preprocessed using *Trimmomatic*[30], version 0.39. We used the *ILLUMINACLIP* and *MINLEN* option with the following arguments, *ILLUMINACLIP:/opt/Trimmomatic0.39/adapters/TruSeq3-PE-2.fa:2:30:10* and *MINLEN:50*. The adapter sequence we used was available in the program folder */opt/Trimmomatic0.39/adapters/*, and the numbers following the file path were the values we defined for the options referred previously, which means, we allowed 2 mismatches per full match while wanting an accuracy of at least 30 between the adapters present in the sample reads and the ones in the file with the adapters, and an accuracy of at least 10 between the sequence of the adapter and the sample. For the *MINLEN* parameter the value we chose was 50, which means we filtered the reads to stay only with the ones that had a minimum length of 50 bases.

The replicas obtained through Ion Torrent sequencing were preprocessed using *SolexaQA*[31], version v.3.1.7.3. We used the *DynamicTrim* and *Lengthsort* algorithms, using the default values of 0.05 for the probability value cutoff in the *DynamicTrim* algorithm, and using the value of 50 for the length cutoff parameter in the Lengthsort algorithm. We didn't use the *Analysis* algorithm since that step had already been performed with *FastQC*[28].

3.2.4. Reads Alignment

After the previous step and before the alignment, we chose to keep only the autosomal chromosomes (1 to 22) in order to focus on the chromosomes of interest and decrease further computational time, using a function we developed to execute this step. This function makes use of *samtools faidx*[86] to select only the twenty-two chromosomes and saves only those in a new FASTA file. Due to chromosome notation compatibility between programs, we also incorporated in this function code to generalize the notation so no problems regarding this matter appeared in subsequent steps.

After that we proceeded with the alignment. We used the *BWA*[32,33] program for the Illumina replicas, and *TMAP*[36] for the Ion Torrent ones. Before the alignment we indexed the reference genome with the same programs used to perform the alignment. In the case of *BWA*, we used the *bwa-mem* algorithm and used the -R argument to specify the header. Regarding *TMAP*, we used the *mapall* option, including all algorithms except the *mapvsw*, and did not use any optional argument.

3.2.5. BAMs processing

As said previously, it is recommended to process the BAM files obtained after the alignment. Therefore, we followed the *GATK* Best practices[38] and used GATK *MarkDuplicates, BaseRecalibrator* and *ApplyBQSR* tools to execute this step for both replicas.

3.2.6. Variant calling

We performed the variant calling with more than one program for both sequencing cases.

Starting with Illumina, we used *Contra*[58] and *VarScan2*[50]. For *Contra*, we used the optional parameters *-p* and *-l*, but apart from that used the default values for the other arguments, such as the number of bins (Default value : 20), minimum number of bases in each target region (Default value: 10) and p-value threshold (Default value: 0.05). Regarding *VarScan2*, we applied every recommended step, except the last one, the merging adjacent segments, due to errors in the program script that were incompatible with our work. We used the default values.

In the case of the Ion Torrent replicates, we used the *GATK* “Sensitively detect copy ratio alterations and allelic segments” tutorial[61], and *VarScan2*[50] as well.

We followed *GATK* tutorial, using most of the times the default or recommended values, except in the *CreateReadCountPanelOfNormals* tool, where specified an optional parameter, *minimum-interval-median-percentile*, and changed its default value of 10.0 to 5.0. This parameter filters genomic intervals which median is above the defined percentile value, keeping only the ones that are above that value. By changing this value to 5.0, we could maintain more information. Asides from that we additionally plotted the copy and allelic ratios segmentation results using the *PlotModeledSegments* tool.

For *VarScan2*, the procedure was the same as the one used with the Illumina replicates.

4. Results

4.1. Literature Review

We searched on PubMed for the mesh terms “exome variant callers” and “analysis software for somatic cells”, and collected 215 articles that were used to perform our literature review. From these, we chose 101 different programs that could potentially integrate our pipeline since they verified the criteria showed in **figure 8**. We analyzed and compared them and chose 9 programs to make part of our pipeline (**Table 4**).

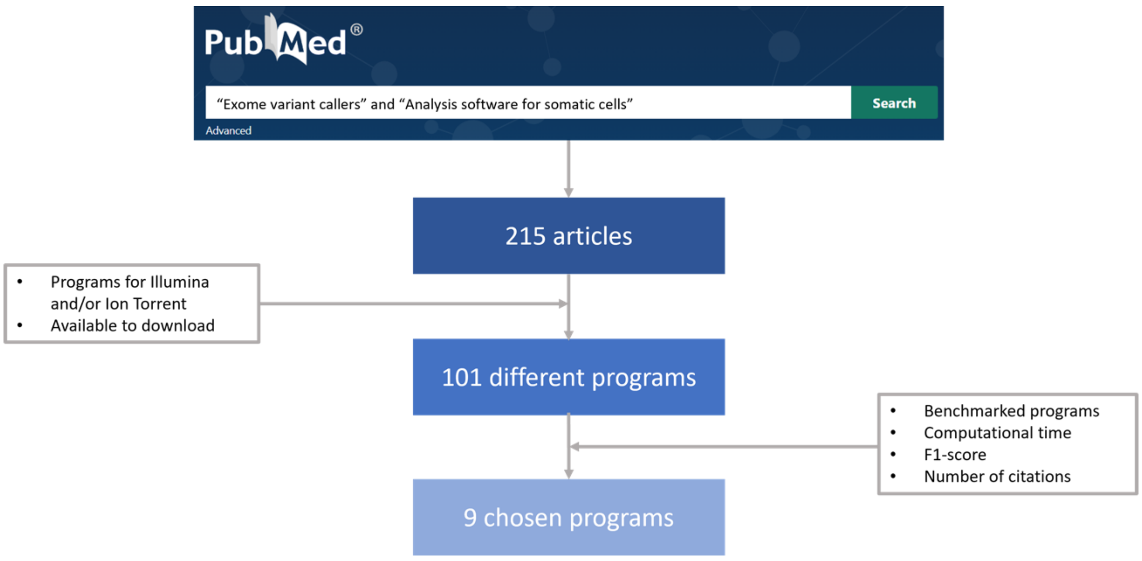


Figure 8 - Literature review process

Table 4 - Chosen programs and reason of choosing

Programs	Source	Reasons for choosing them
FastQC[28]	S., A. (2010). FastQC: a quality control tool for high throughput sequence data. http://www.bioinformatics.babraham.ac.uk/projects/fastqc	Good performance Number of citations Most used worldwide

Trimmomatic[30]	Anthony M. Bolger, Marc Lohse, Bjoern Usadel, Trimmomatic: a flexible trimmer for Illumina sequence data, Bioinformatics, Volume 30, Issue 15, August 2014, Pages 2114–2120, https://doi.org/10.1093/bioinformatics/btu170	Benchmarking Good performance Number of citations
SolexaQA[31]	Cox, M. P., Peterson, D. A., & Biggs, P. J. (2010). SolexaQA: At-a-glance quality assessment of Illumina second-generation sequencing data. BMC Bioinformatics, 11, 485. https://doi.org/10.1186/1471-2105-11-485	Good performance Works with Ion Torrent files
BWA[32,33]	Li, H., & Durbin, R. (2009). Fast and accurate short read alignment with Burrows-Wheeler transform. Bioinformatics, 25(14), 1754-1760. https://doi.org/10.1093/bioinformatics/btp324 Li, H., & Durbin, R. (2010). Fast and accurate long-read alignment with Burrows-Wheeler transform. Bioinformatics, 26(5), 589-595. https://doi.org/10.1093/bioinformatics/btp698	Computational time Good performance Number of citations Most used Worldwide
TMAP[36]	Caboche, S., Audebert, C., Lemoine, Y., & Hot, D. (2014). Comparison of mapping algorithms used in high-throughput sequencing: application to Ion Torrent data. BMC Genomics, 15, 264. https://doi.org/10.1186/1471-2164-15-264	Specifically designed for Ion Torrent
Contra[58]	Li, J., Lupat, R., Amarasinghe, K. C., Thompson, E. R., Doyle, M. A., Ryland, G. L., Tothill, R. W., Halgamuge, S. K., Campbell, I. G., & Gorringe, K. L. (2012). CONTRA: copy number analysis for targeted resequencing. Bioinformatics, 28(10), 1307-1313. https://doi.org/10.1093/bioinformatics/bts1467	Detection of small and large CNVs
GATK[61]	https://gatk.broadinstitute.org/hc/en-us/articles/360035531092--How-to-part-I-Sensitively-detect-copy-ratio-alterations-and-allelic-segments	GATK tools perform very well and are used worlwide
VarScan2[50]	Koboldt, D. C., Zhang, Q., Larson, D. E., Shen, D., McLellan, M. D., Lin, L., Miller, C. A., Mardis, E. R., Ding, L., & Wilson, R. K. (2012). VarScan 2: somatic mutation and copy number alteration discovery in cancer by exome sequencing. Genome Res, 22(3), 568-576. https://doi.org/10.1101/gr.129684.111	F1-Score Number of citations

4.2. Quality Control

The quality control step carried out with *FastQC* reported adaptor contamination in the Illumina replicates (**Fig.9 and 10**), and the presence of low-quality reads in the 3´end of the Ion Torrent replicates (**Fig. 11**). The results obtained with *FastQC* are displayed on **table 5**.

Table 5 - Reads quality before the processing step

Sequencer	Samples	Read	Total reads	Poor quality reads	Mean read length	Presence of adapters
Illumina	Normal sample (SRR7890845)	Read 1	57058265	0	126	Yes
		Read 2	57058265	0	126	Yes
	Tumor sample (SRR7890844)	Read 1	54548549	0	126	Yes
		Read 2	54548549	0	126	Yes
Ion Torrent	Normal sample (SRR8955981)	-	82434896	0	25-353	No
	Tumor sample (SRR8955982)	-	66834918	0	25-355	No

Since Illumina replicates are paired-end, the number of reads per sample is the summation of the number of reads per reads. Therefore, the total number of reads for the normal sample of Illumina is 114116530 reads, and for the tumor sample is 109097098 reads. As we can observe, Ion Torrent replicates had a smaller number of reads per file when compared with the Illumina ones, for both the normal and tumor sample. The mean sequence length of the Illumina replicates was constant for both samples, whereas the replicates that came from Ion Torrent varied greatly in their sequence length, going to the encounter of what is stated in the literature[85].

.

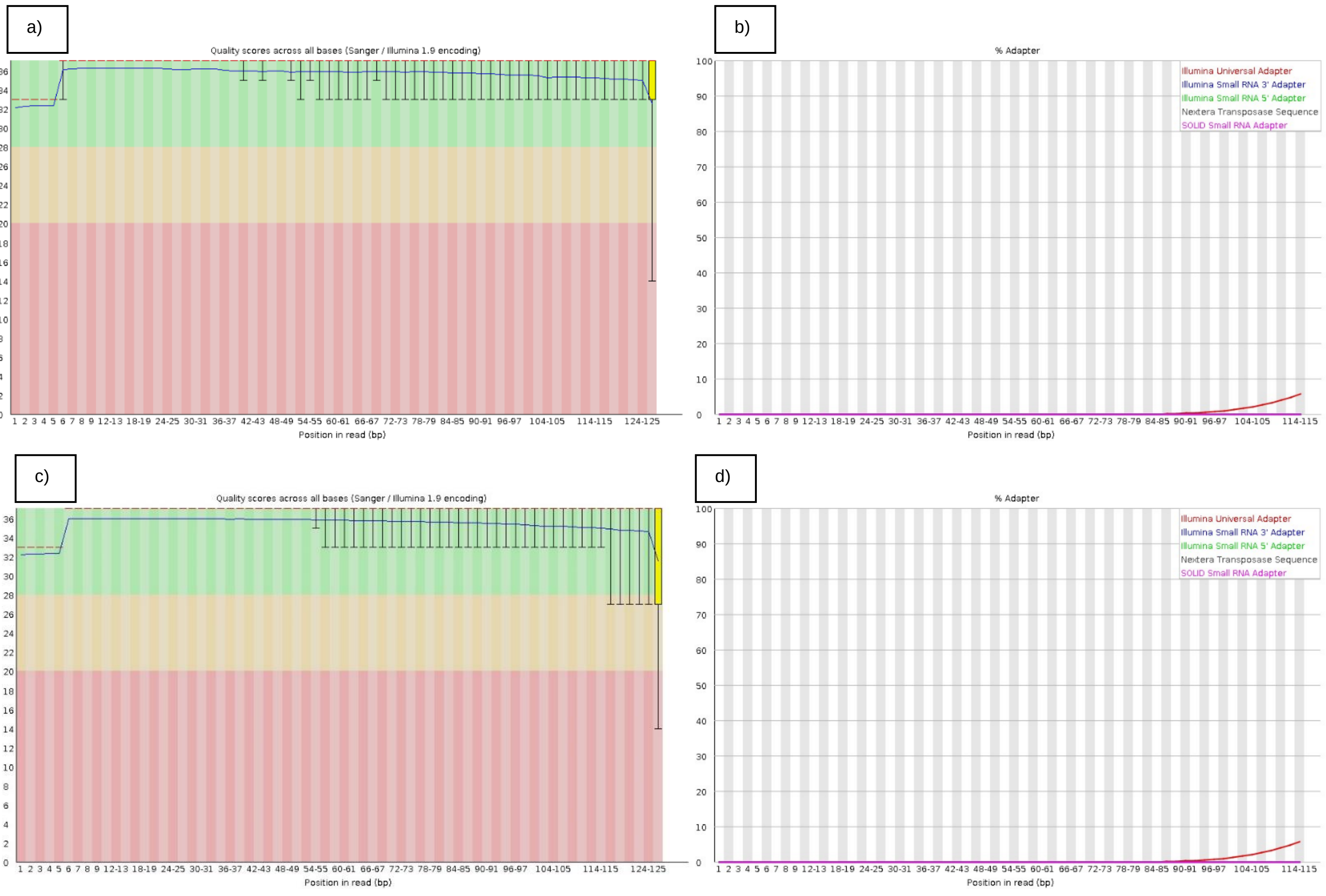


Figure 9 - Per base quality and adapter content of the Illumina replicate of the normal sample before reads processing. a) and b) correspond to read 1, and c) and d) to read 2

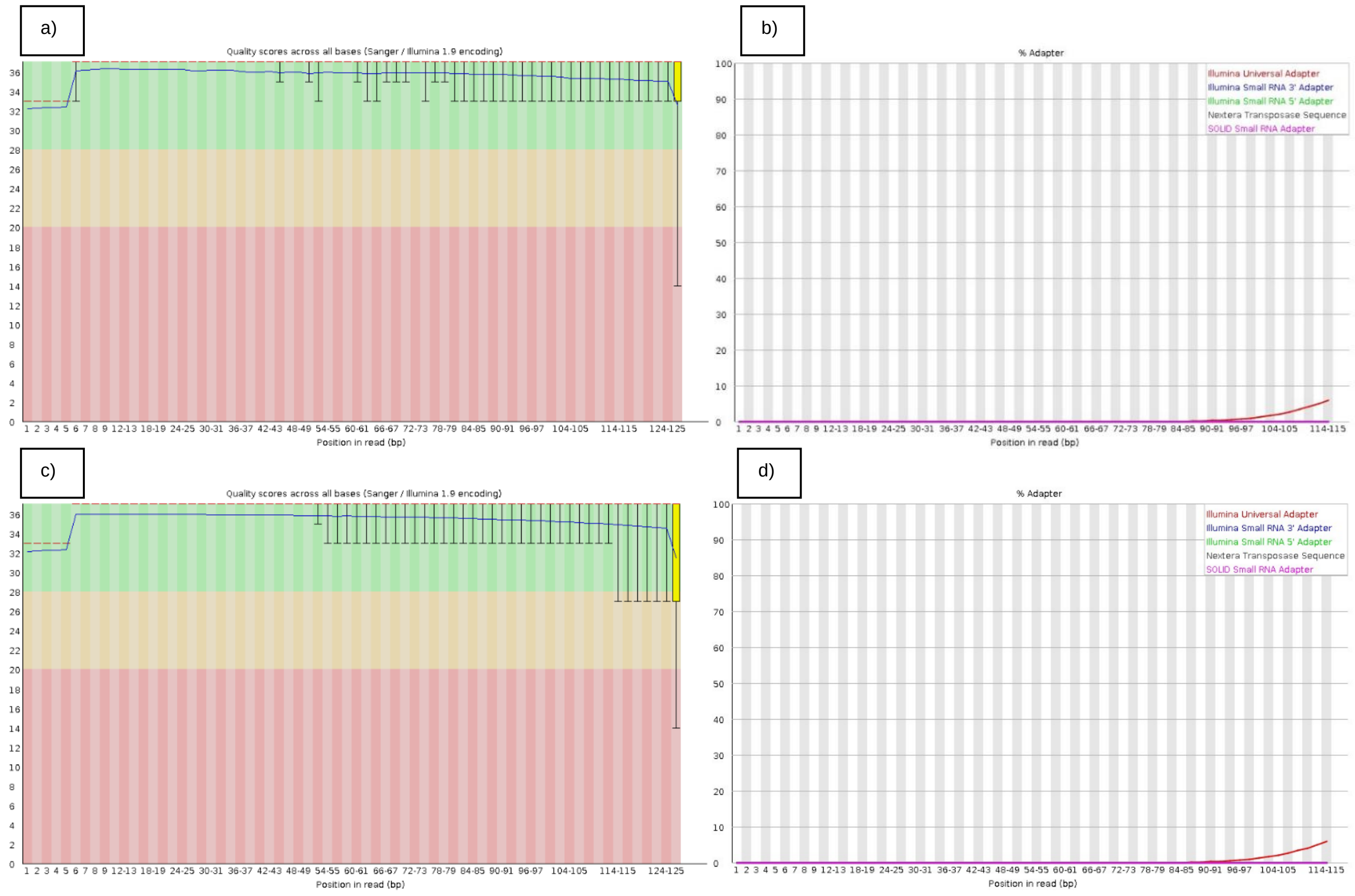


Figure 10 - Per base quality and adapter content of the Illumina replicate of the tumour sample before reads processing. a) and b) correspond to read 1, and c) and d) to read 2

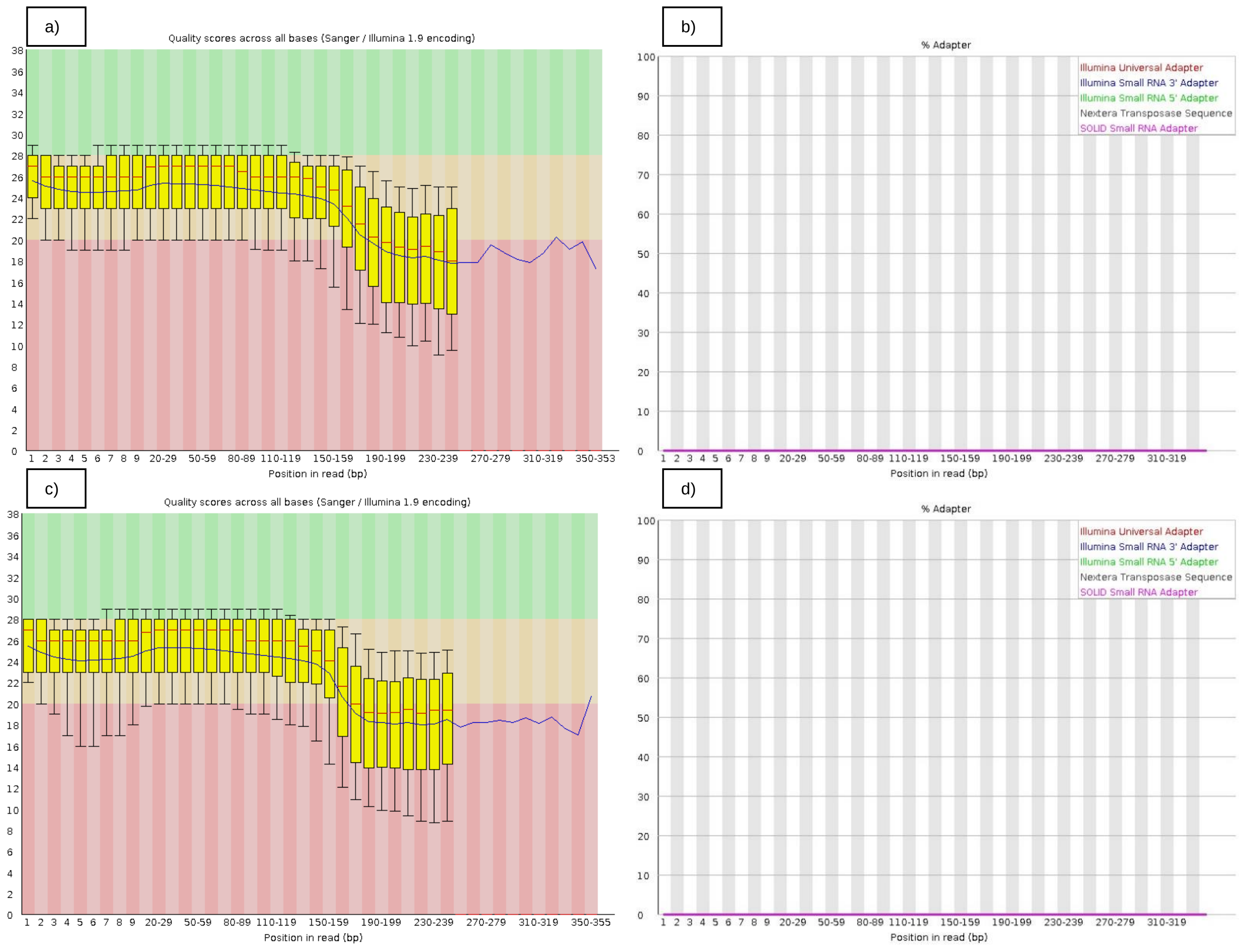


Figure 11 - Per base quality and adapter content of the Ion Torrent replicates before reads processing. a) and b) correspond to the normal sample, and c) and d) to the tumour sample

4.3. Reads Processing

Since Illumina reads had adapter content, we removed the specific adapters of the sequencer used and filtered the reads in order to keep only the ones whose length was equal or higher than 25 (**Fig. 12 and 13**). With regard to the Ion Torrent replicas, we removed the reads whose length was below 25. In both cases we decided to filter reads to stay with the ones that were equal or higher than 25 because we are looking for SVs, which are known to be of at least 50 bases, however, it can happen that there are reads with 25 bps whose mate can have an insert size bigger, and therefore be a SV. So, by using this value it's not only possible to detect SVs with 50 bases but also detect this cases. Besides that, we also tackled the problem of the low-quality reads of the Ion Torrent replicates as well simply by filtering out the reads as previously said (**Fig. 14**).

Table 6 - Reads quality after the processing step

Sequencer	Samples	Total reads	Mean read length	Percentage of reads lost
Illumina	Normal sample (SRR7890845)	52172914	28-126	~ 8.56%
		52172914	25-126	~ 8.56%
	Tumor sample (SRR7890844)	49793638	26-126	~ 8.72%
		49793638	25-126	~ 8.72%
Ion Torrent	Normal sample (SRR8955981)	80859855	25-207	~ 1.91%
	Tumor sample (SRR8955982)	65011616	25-195	~ 2.73%

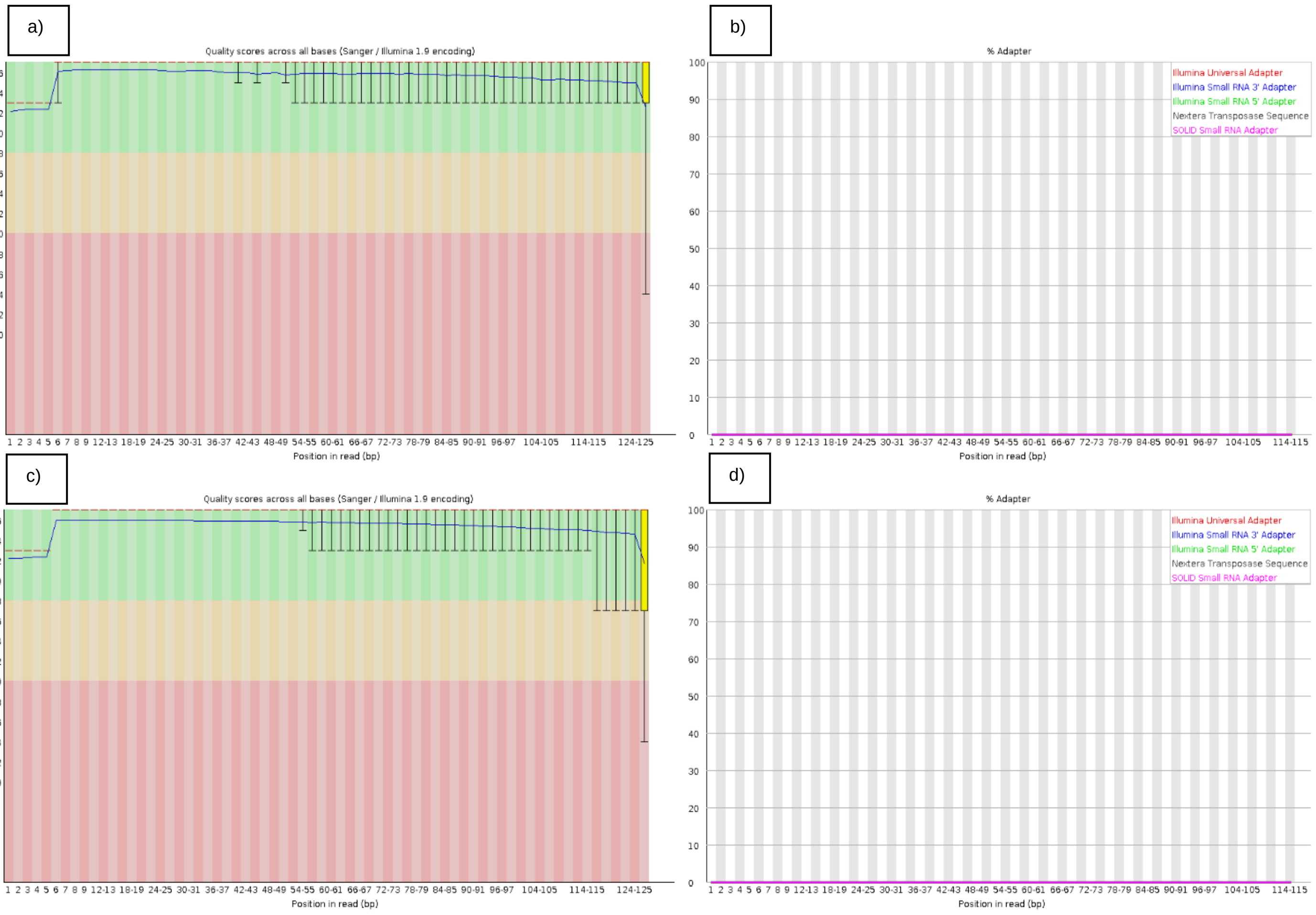


Figure 12 - Per base quality and adapter content of the Illumina replicate of the normal sample after reads processing. a) and b) correspond to read 1, and c) and d) to read 2

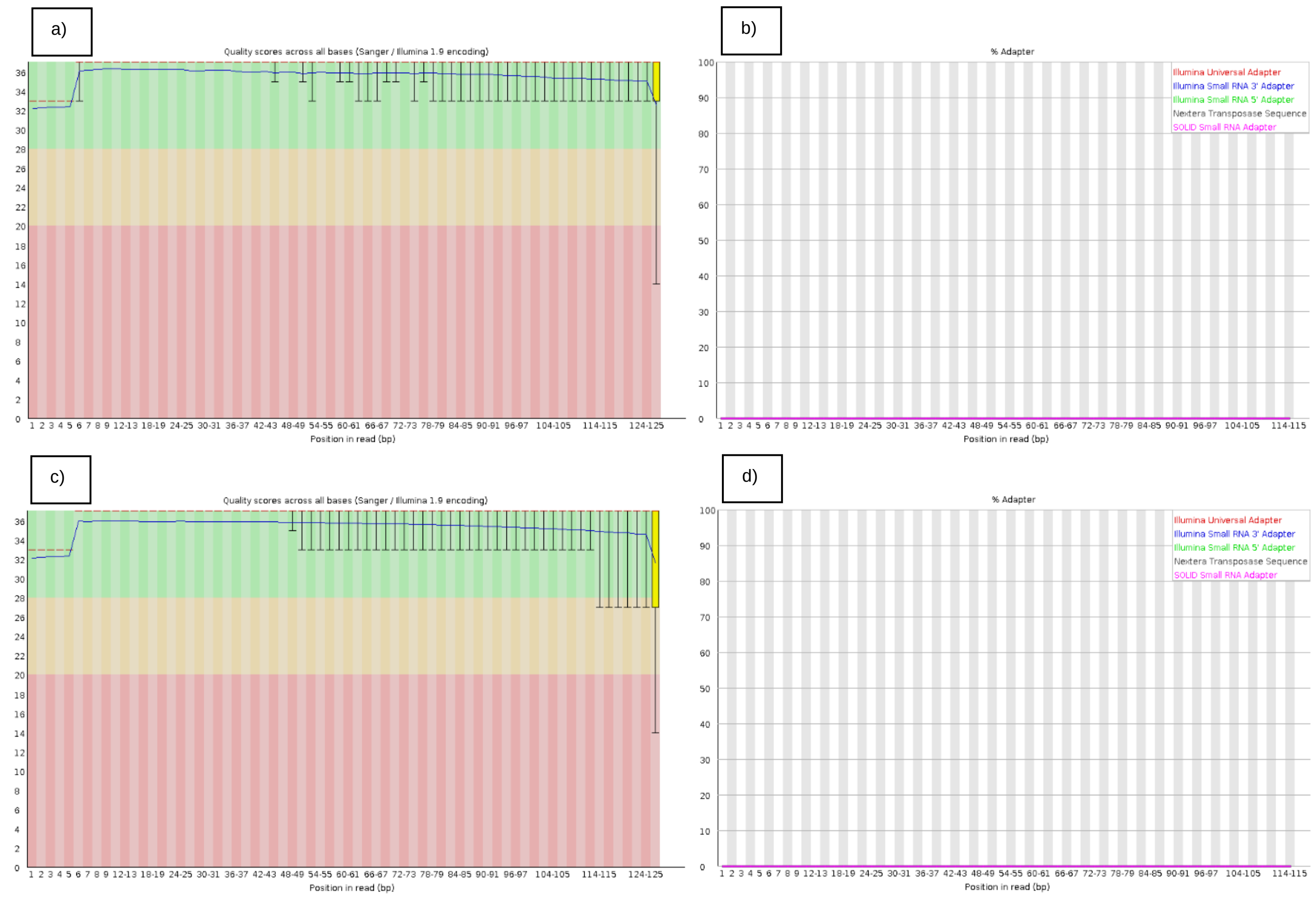


Figure 13 - Per base quality and adapter content of the Illumina replicate of the tumour sample after reads processing. a) and b) correspond to read 1, and c) and d) to read 2

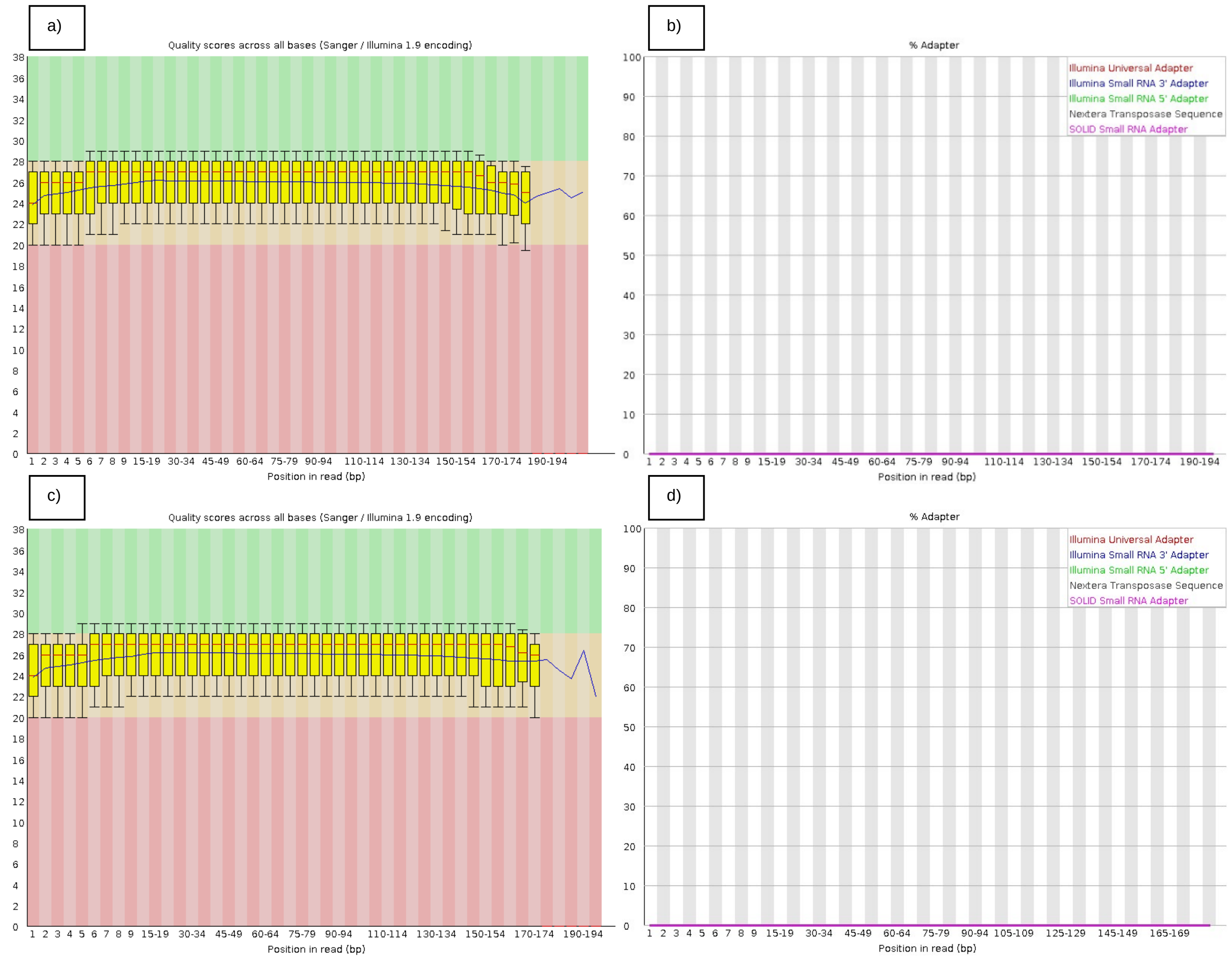


Figure 14 - Per base quality and adapter content of the Ion Torrent replicates after reads processing. a) and b) correspond to the normal sample, and c) and d) to the tumour sample

4.4. Alignment and BAMs processing

After the alignment and BAMs processing, statistics were obtained using *samtools flagstat* and are displayed on **table 7**. With respect to the Illumina samples, we achieved a 98.52% alignment for the normal sample and 98.68% for the tumor sample, whereas for the Ion Torrent samples we achieved 98.81% alignment for the normal sample and 98.71% for the tumor sample.

Table 7 - Samtools flagstat report after alignment and BAMs processing

Sequencer	Samples	Total reads before alignment	Mapped reads (% of alignment)	Number of reads marked as duplicated
Illumina	Normal sample (SRR7890845)	104534734	102987576 (98.52%)	3543402
	Tumor sample (SRR7890844)	99761492	98440858 (98.68%)	3097110
Ion Torrent	Normal sample (SRR8955981)	82434896	81452727 (98.81%)	36987065
	Tumor sample (SRR8955982)	66834918	65970458 (98.71%)	28772928

4.5. Variant calling

The number of variants obtained from *Contra* and *VarScan2* for Illumina, and from *GATK* and *VarScan2* for Ion Torrent is showed in **Figure 15**. As it's possible to see, Contra returned much more variants than *VarScan2* for Illumina, around 278.32% more variants. With respect to Ion Torrent, *VarScan2* returned more variants than *GATK*, but the difference between them is not as big as the one seen in Illumina, with *VarScan2* calling 2.23% more variants than *GATK*.

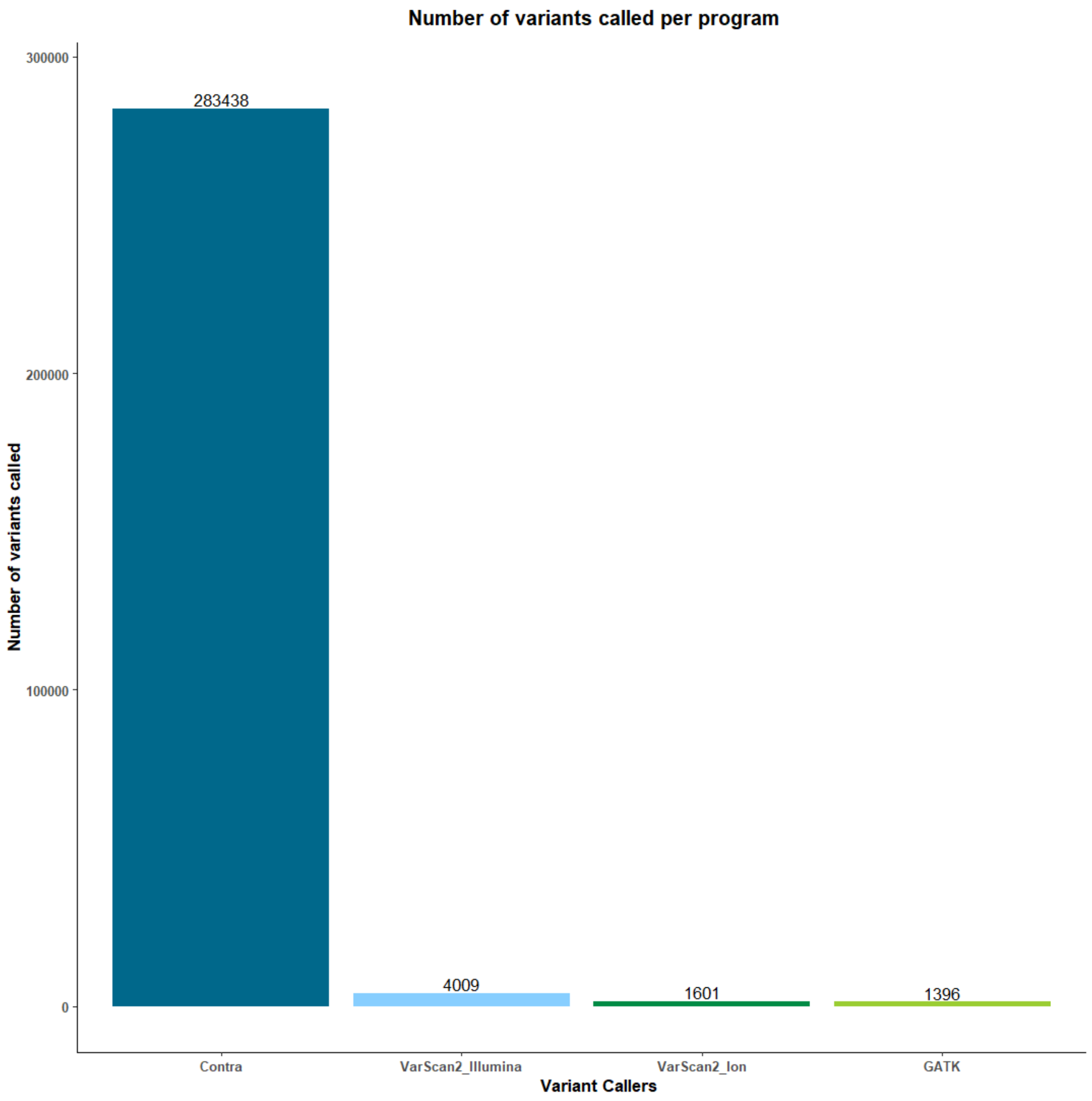


Figure 15 - Number of variants called per program

In **Section A.2** of the Supplementary Information, it’s possible to observe that the distribution of the variants per chromosome along all variant callers was not the same, despite the total number of variants called. The majority of the chromosomes occupies the same position in both callers for a specific replicate, but there are some discordances, nonetheless. For example, chromosome 1, occupies the same position in both callers for the Illumina replicate, however chromosome 3 is the 3rd chromosome with the most called variants in VarScan2, but in Contra is the 7th one (**Section A.2.1**).

The log ratio distribution for each caller is shown next (**Fig.16 to 19**).

- **Contra**

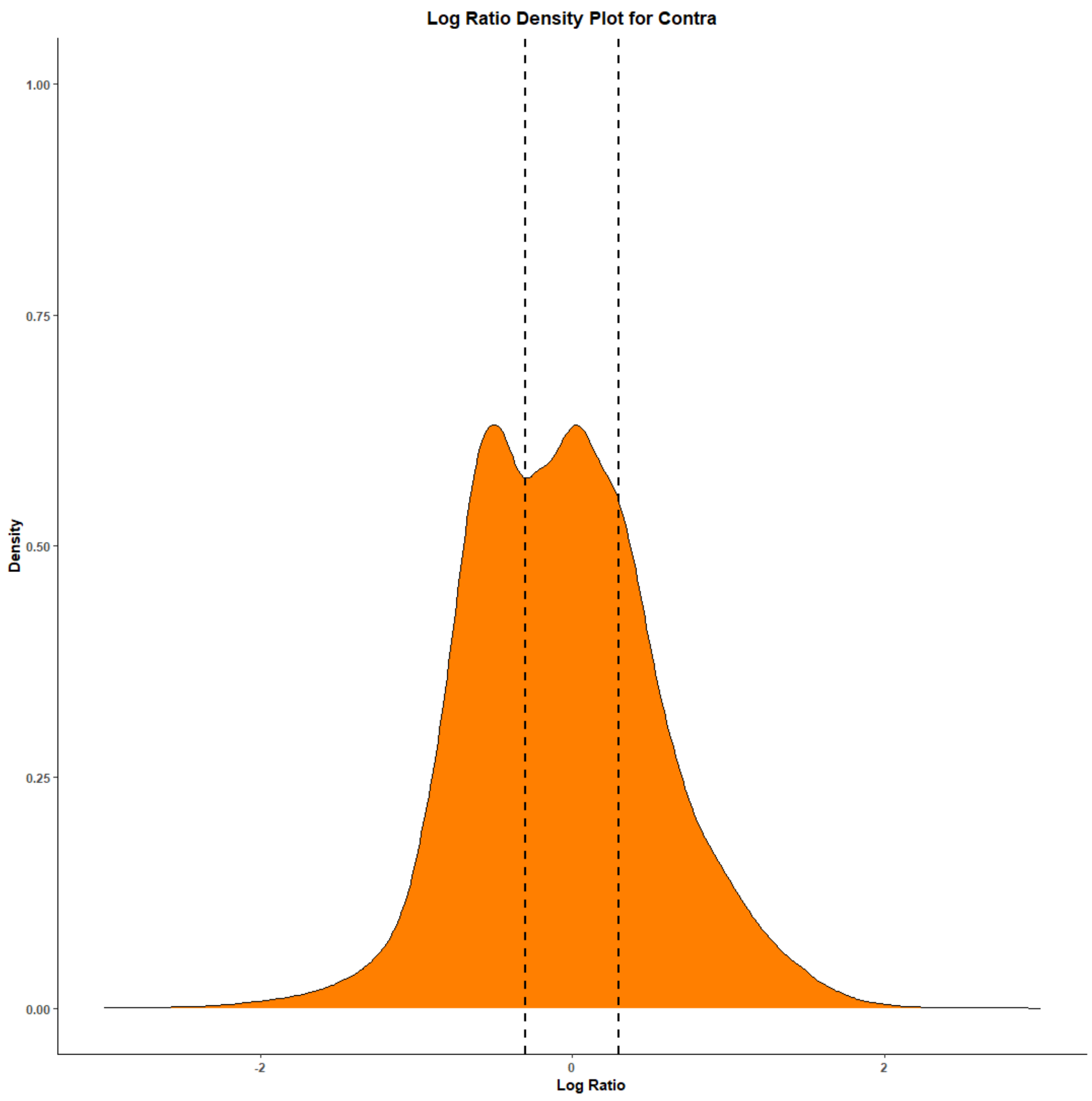


Figure 16 - Distribution of the Log Ratio in Contra

Contra classifies CNV gains when the log-ratio is above 0.3 or losses if it is below – 0.3[58]. As it's possible to see from Fig.16, there are two spikes with similar density, one around - 0.5 and the other around 0. This means we have a greater number of variants with log-ratio values of -0.5 and 0, and therefore there are more SVs classified as losses and neutral variants, with low density of variants classified as gains.

- **VarScan2 (Illumina)**

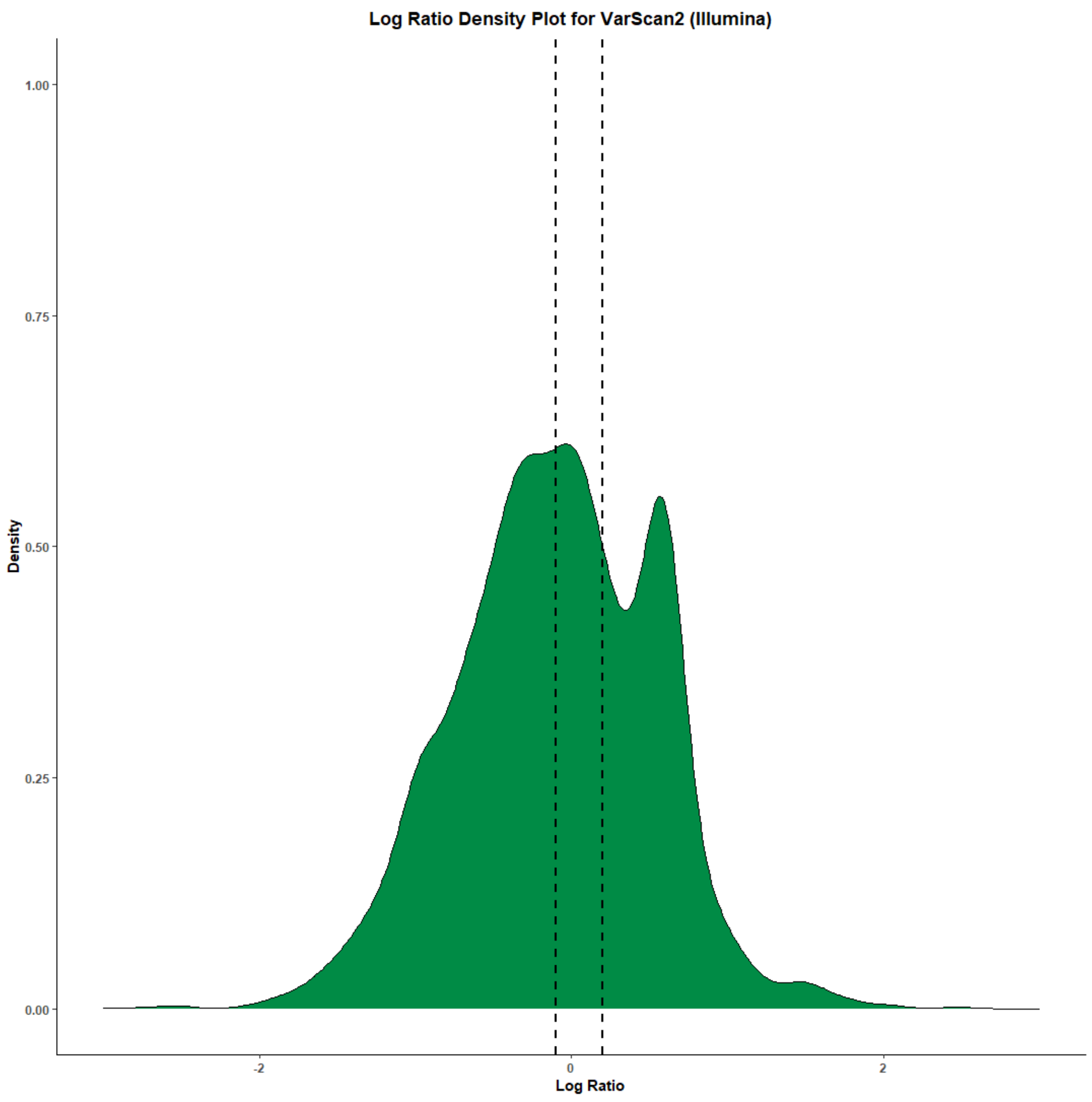


Figure 17 - Distribution of the Log Ratio in VarScan2 (Illumina)

For VarScan2, values higher than 0.20 of log ratio represent amplifications, and values below -0.10 represent deletions[50]. As it's possible to see from **Fig.17**, there is a higher density of variants around the threshold of -0.10, and a spike of density of positive values around 0.75 approximately. This means that there is a high density of variants that are considered losses and neutral. However, there is also a moderate density of variants classified as gains.

- **GATK**

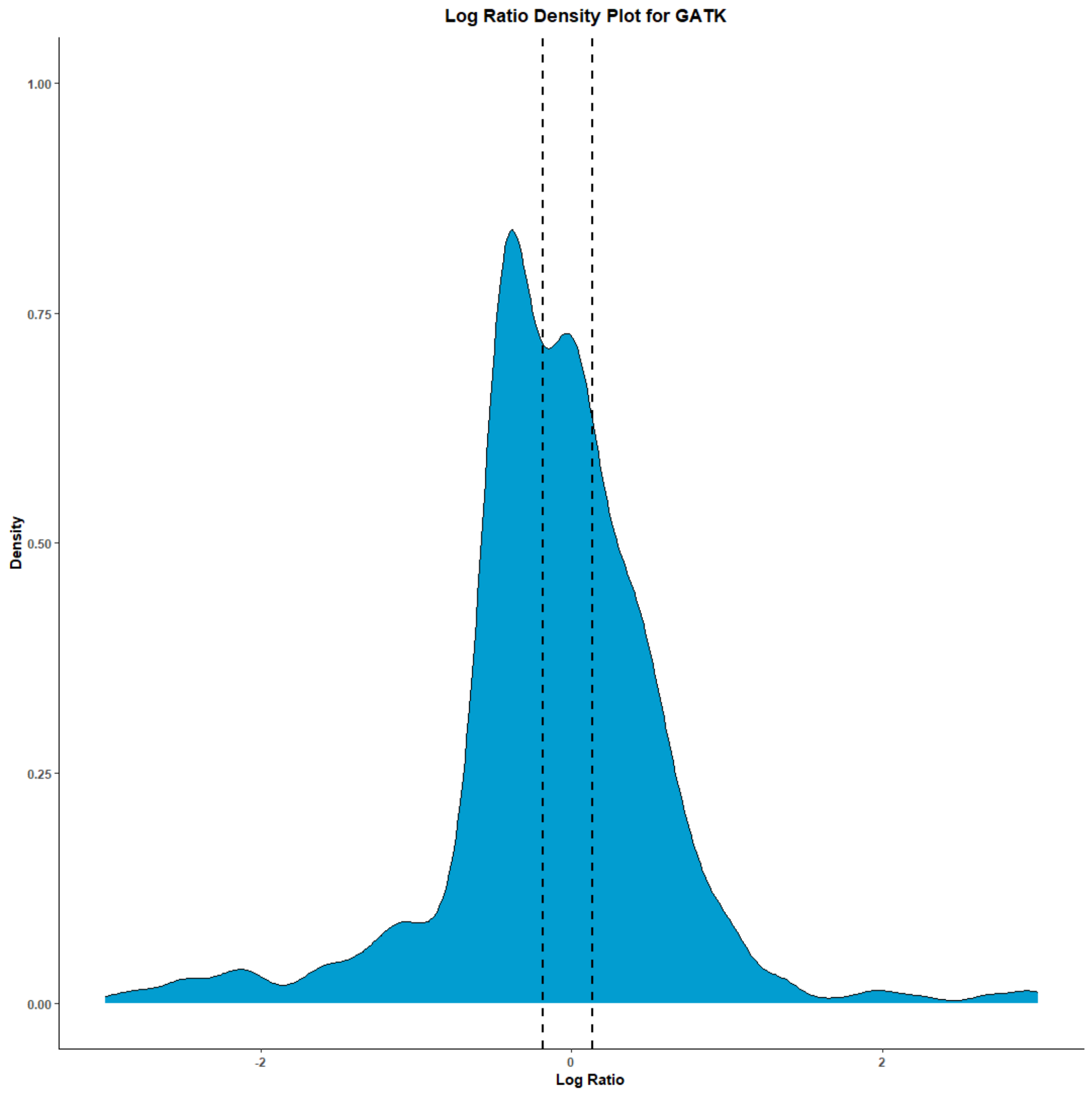


Figure 18 - Distribution of the Log Ratio in GATK

GATK considers losses when log-ratio values are below - 0.183 and considers gains when the log-ratio values are above 0.133 (**Fig. 18**). As it's possible to see there is one spike on the loss area, around the -0.20, and a smaller one in the neutral area, followed by a decrease of density for values above the upper threshold of 0.133. This shows that there are more variants considered as losses and neutral then gains.

- **VarScan2 (Ion Torrent)**

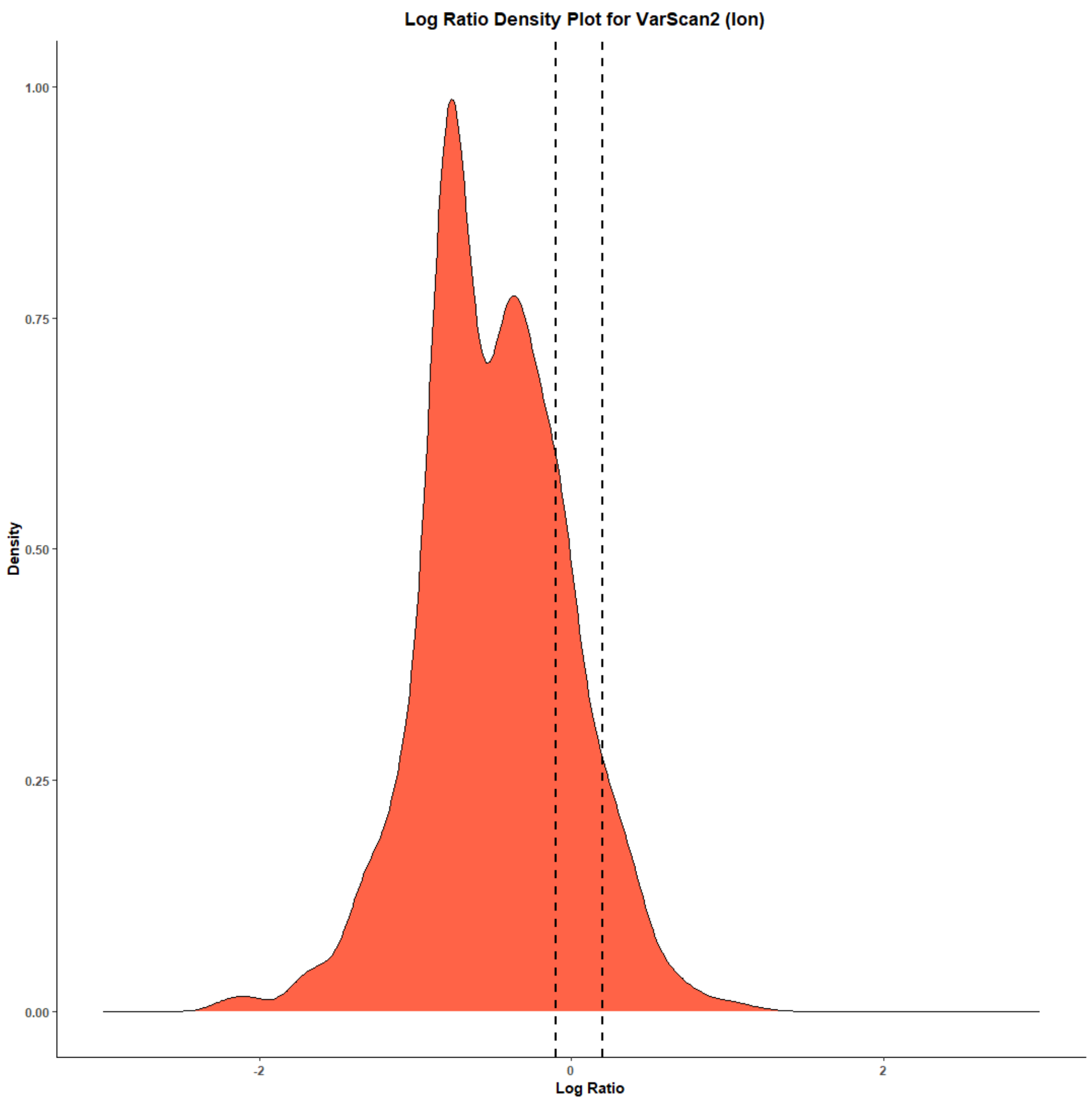


Figure 19 - Distribution of the Log Ratio in VarScan2 (Ion Torrent)

For the Ion Torren replicates, VarScan2 shows two spikes of density below the lower threshold, meaning once again, that the majority of variants is classified as a loss. It is also possible to see there is a huge decrease regarding the density of variants that are neutral or gains. (**Fig. 19**).

4.6. Merging of variant caller’s results

After merging the files of both callers for each of the replicates we obtained around 1.34% (3841 variants) that were called for both callers in Illumina replicates. Regarding the Ion Torrent replicates we obtained 45.78% (1372 variants) of variants called by both callers. When comparing the variants called by Contra with the ones called by the Ion Torrent variant callers, GATK and VarScan2, the percentage of variants that were called by both of them was 0.49% (1386 variants) and 0.55% (1579 variants) respectively. However, when comparing the VarScan2 results for the Illumina replicates with the GATK and VarScan2 results for Ion Torrent, the percentage of variants called by both of them was 25.81% (1395 variants) and 28.54% (1601 variants) respectively.

Table 8 - Intersection between callers

		Illumina		**Ion Torrent**	
		Contra	**VarScan2**	**GATK**	**VarScan2**
Illumina	**Contra**				
	VarScan2	~ 1.34%			
Ion Torrent	**GATK**	~ 0.49%	~ 25.81%		
	VarScan2	~0.55%	~ 28.54%	~ 45.78%	

5. Discussion

Variant identification and characterization are a field of high importance among researchers. It relies on the use of many programs to obtain information from the data that comes out of the sequencer, with special focus on the programs used to perform the variant calling. Literature advises to use more than one variant caller, and therefore that was what we did, by combining the results of two different callers for CNV identification[24].

With the first step in the workflow, the literature review, we noticed that the majority of programs developed for different tasks of variant calling pipelines are for data from Illumina sequencers. A possible explanation for this fact is that Illumina is the most used sequencer worldwide[88]. Finding programs for Ion Torrent proved to be a challenge due to this. We also noticed that a large part of the programs described in literature have not been updated or received bug fixes in recent years, making them obsolete and hard to work with. We experienced this with some programs that were initially part of our pipeline, but due to bugs or incompatibilities between the programs dependencies version and the one present in our system, we had to discard them. Even with some of the programs we used, we had to resort to virtual environments to use them due to the dependencies issue we mentioned, causing us to change the environment constantly between the steps of the pipeline.

When we started to work with the data, we identified some issues that were already described in the literature (Section 1.2). Illumina replicates were contaminated with adapters, and in the case of the Ion Torrent replicates, their quality decreased through the 3’end of the reads. The programs we chose to process the reads and get rid of these problems performed very well (Section 4.3). These also helped us achieve high alignment percentages, with above 98% of the reads being aligned to the reference[90].

Regarding the variant calling we obtained good results using our methods and most of the time, using the default values of the programs, however, this is not valid for all cases. We tried to change the default values, but the best results were obtained proceeding the way we did. For example, in the case of Contra, we tried to change the default values to see if we could improve the results. We decided to change the number of bins to group the regions, but no improvement was verified when using less or more than 20 bins, the default value (**Fig. 22**).

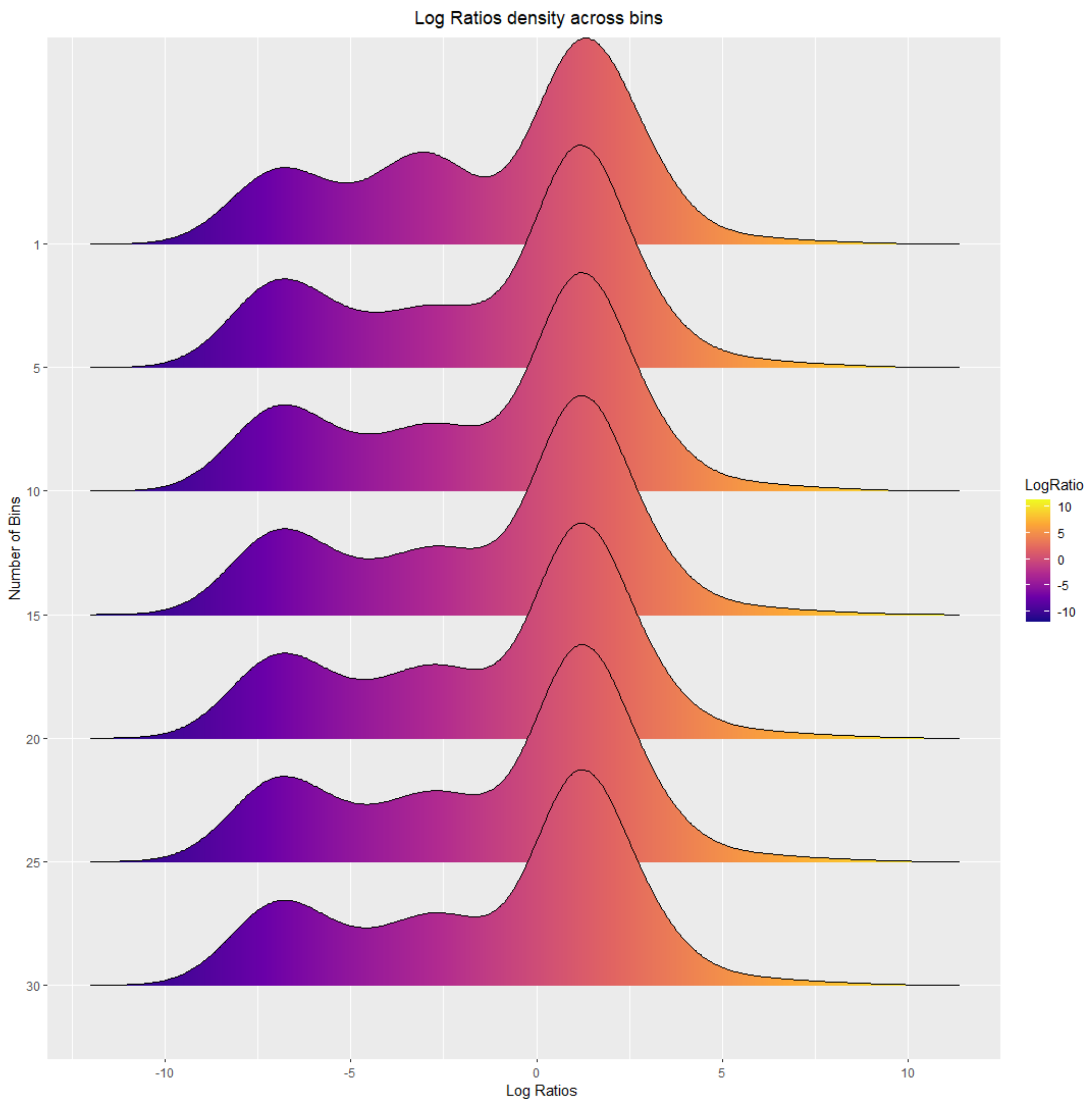


Figure 20 - Log Ratio distribution across number of bins

As said previously, the use of more than one variant caller increases the confidence on the calls, therefore, in our case, we have more confidence in the variants that were called by the two variants than the ones who were only called by one.

6. Conclusion

6.1. Conclusions

Our pipeline achieved a high number of calls when using the combination of two variant callers for both sequencer replicates. For Illumina replicates, *Contra* was the caller that returned more variants, while for Ion Torrent it was *VarScan2*. Nonetheless, and as it was possible to see previously, the callers have different density of variants distributed across the log-ratio scales, apart from having different log-ratio thresholds. This highlights the importance of using more than one variant caller, so it is possible to achieve the maximum insights possible from the data, since different callers use different algorithms that lead to some specific variants called only by that caller, and that can be of great interest for researchers. Apart from that, the choice of using different and more specific programs for the reads pre-processing and alignment steps, previous to variant calling, allowed us to achieve good filtering results and high percentage of alignment between the read and the reference genome.

We expect that by improving the results of this important method for variant discovery, researchers can obtain more precise insights and make better choices for treatments and clinical diagnostics.

6.2. Limitations

During this work we experience first-hand the difficulty within the use of some programs. Even though some of them are still available online, most have not been updated or received corrections for some years, what caused some problems with their dependencies and their use when integrated with other programs. Besides that, variant calling methods available nowadays are still prone to errors, due to NGS library preparation and lack of standardization of this method. SV's detections are even more difficult to detect since variant callers designed for this specific type of variants make use of short reads that limit these detection[5].

Regarding the sequencers used to obtain WES data, each year there are new methods or sequencers available to perform this task, which makes it difficult to choose the right platform in which to sequence the data. This also influences the type of programs to choose, since programs are developed according to the provenience of the data regarding these matters.

Lastly, this pipeline is prone to changes, since the programs that compose it might suffer alterations that could lead to malfunctioning of some steps of the pipeline.

6.3. Future Work

To improve the work developed in this dissertation, we can add in the future programs to detect other types of variants, such as SNV/indels. Additional studies using benchmarked datasets, such as the ones available in the Genome in a Bottle Consortium (GIAB), is something we plan to do in the future since it can help tune some parameters regarding the pipeline. With the continue release of more programs, the variant callers used here might be changed if a better combination of programs is found, besides adding more than two variant callers per sequencer.

Furthermore, accepting data from other sequencers, and therefore implement other programs specific to that sequencer can be an improvement to consider.

References

1. Alberts, B. (2022). Molecular biology of the cell (Seventh edition. ed.). W. W. Norton & Company.
2. Ku, C. S., Loy, E. Y., Salim, A., Pawitan, Y., & Chia, K. S. (2010). The discovery of human genetic variations and their use as disease markers: past, present and future. Journal of Human Genetics, 55(7), 403-415. https://doi.org/10.1038/jhg.2010.55
3. The 1000 Genomes Project Consortium et al. A global reference for human genetic variation. Nature 526, 68–74 (2015).
4. Karki, R., Pandya, D., Elston, R. C., & Ferlini, C. (2015). Defining "mutation" and "polymorphism" in the era of personal genomics. BMC Med Genomics, 8, 37. https://doi.org/10.1186/s12920-015-0115-z
5. Mahmoud, M., Gobet, N., Cruz-Davalos, D. I., Mounier, N., Dessimoz, C., & Sedlazeck, F. J. (2019). Structural variant calling: the long and the short of it. Genome Biol, 20(1), 246. https://doi.org/10.1186/s13059-019-1828-7
6. Chen, S., Francioli, L. C., Goodrich, J. K., Collins, R. L., Kanai, M., Wang, Q., Alföldi, J., Watts, N. A., Vittal, C., Gauthier, L. D., Poterba, T., Wilson, M. W., Tarasova, Y., Phu, W., Yohannes, M. T., Koenig, Z., Farjoun, Y., Banks, E., Donnelly, S., . . . Karczewski, K. J. (2022). A genome-wide mutational constraint map quantified from variation in 76,156 human genomes. bioRxiv, 2022.2003.2020.485034. https://doi.org/10.1101/2022.03.20.485034
7. Botstein, D., & Risch, N. (2003). Discovering genotypes underlying human phenotypes: past successes for mendelian disease, future approaches for complex disease. Nat Genet, 33 Suppl, 228-237. https://doi.org/10.1038/ng1090
8. Boyle, E. A., Li, Y. I., & Pritchard, J. K. (2017). An Expanded View of Complex Traits: From Polygenic to Omnigenic. Cell, 169(7), 1177-1186. https://doi.org/10.1016/j.cell.2017.05.038
9. Vijg, J. (2014). Somatic mutations, genome mosaicism, cancer and aging. Curr Opin Genet Dev, 26, 141-149. https://doi.org/10.1016/j.gde.2014.04.002
10. Morley, A. A. (1995). The somatic mutation theory of ageing. Mutat Res, 338(1-6), 19-23. https://doi.org/10.1016/0921-8734(95)00007-s
11. Luzzatto, L. (2011). Somatic mutations in cancer development. Environ Health, 10 Suppl 1(Suppl 1), S12. https://doi.org/10.1186/1476-069X-10-S1-S12
12. Wright, A.F. (2005). Genetic Variation: Polymorphisms and Mutations. In eLS, (Ed.). https://doi.org/10.1038/npg.els.0005005

13. Vogelstein, B., Papadopoulos, N., Velculescu, V. E., Zhou, S., Diaz, L. A., Jr, & Kinzler, K. W. (2013). Cancer genome landscapes. Science (New York, N.Y.), 339(6127), 1546–1558. https://doi.org/10.1126/science.1235122

14. Roth, J. R. (1974). FRAMESHIFT MUTATIONS. Annual Review of Genetics, 8(1), 319-346. https://doi.org/10.1146/annurev.ge.08.120174.001535

15. van Dijk, E. L., Auger, H., Jaszczyszyn, Y., & Thermes, C. (2014). Ten years of next-generation sequencing technology. Trends Genet, 30(9), 418-426. https://doi.org/10.1016/j.tig.2014.07.001

16. Bentley, D. R., Balasubramanian, S., Swerdlow, H. P., Smith, G. P., Milton, J., Brown, C. G., Hall, K. P., Evers, D. J., Barnes, C. L., Bignell, H. R., Boutell, J. M., Bryant, J., Carter, R. J., Keira Cheetham, R., Cox, A. J., Ellis, D. J., Flatbush, M. R., Gormley, N. A., Humphray, S. J., . . . Smith, A. J. (2008). Accurate whole human genome sequencing using reversible terminator chemistry. Nature, 456(7218), 53-59. https://doi.org/10.1038/nature07517

17. Ambardar, S., Gupta, R., Trakroo, D., Lal, R., & Vakhlu, J. (2016). High Throughput Sequencing: An Overview of Sequencing Chemistry. Indian J Microbiol, 56(4), 394-404. https://doi.org/10.1007/s12088-016-0606-4

18. Alfares, A., Aloraini, T., Subaie, L. A., Alissa, A., Qudsi, A. A., Alahmad, A., Mutairi, F. A., Alswaid, A., Alothaim, A., Eyaid, W., Albalwi, M., Alturki, S., & Alfadhel, M. (2018). Whole-genome sequencing offers additional but limited clinical utility compared with reanalysis of whole-exome sequencing. Genet Med, 20(11), 1328-1333. https://doi.org/10.1038/gim.2018.41

19. Drmanac, R., Sparks, A. B., Callow, M. J., Halpern, A. L., Burns, N. L., Kermani, B. G., Carnevali, P., Nazarenko, I., Nilsen, G. B., Yeung, G., Dahl, F., Fernandez, A., Staker, B., Pant, K. P., Baccash, J., Borcherding, A. P., Brownley, A., Cedeno, R., Chen, L., . . . Reid, C. A. (2010). Human genome sequencing using unchained base reads on self-assembling DNA nanoarrays. Science, 327(5961), 78-81. https://doi.org/10.1126/science.1181498

20. Majewski, J., Schwartzentruber, J., Lalonde, E., Montpetit, A., & Jabado, N. (2011). What can exome sequencing do for you? J Med Genet, 48(9), 580-589. https://doi.org/10.1136/jmedgenet-2011-100223

21. Rabbani, B., Tekin, M., & Mahdieh, N. (2014). The promise of whole-exome sequencing in medical genetics. J Hum Genet, 59(1), 5-15. https://doi.org/10.1038/jhg.2013.114

22. Wetterstrand KA. DNA Sequencing Costs: Data from the NHGRI Genome Sequencing Program (GSP) Available at: www.genome.gov/sequencingcostsdata

23. Muñoz-Barrera, A., Rubio-Rodríguez, L. A., Díaz-de Usera, A., Jáspez, D., Lorenzo-Salazar, J. M., González-Montelongo, R., García-Olivares, V., & Flores, C. (2022). From Samples to Germline and Somatic Sequence Variation: A Focus on Next-Generation Sequencing in Melanoma Research. Life (Basel, Switzerland), 12(11), 1939. https://doi.org/10.3390/life12111939

24. Sims, D., Sudbery, I., Ilott, N. E., Heger, A., & Ponting, C. P. (2014). Sequencing depth and coverage: key considerations in genomic analyses. Nature reviews. Genetics, 15(2), 121–132. https://doi.org/10.1038/nrg3642

25. Koboldt, D. C. (2020). Best practices for variant calling in clinical sequencing. Genome Med, 12(1), 91. https://doi.org/10.1186/s13073-020-00791-w

26. Merino, G. A., Fresno, C., Netto, F., Netto, E. D., Pratto, L., & Fernandez, E. A. (2016). The impact of quality control in RNA-seq experiments. 20th Argentinean Bioengineering Society Congress (Xx Argentine Bioengineering Congress and Ix Conference of Clinical Engineering), (Sabi 2015), 705. https://doi.org/Artn 01200310.1088/1742-6596/705/1/012003

27. Trivedil, U. H., Cezard, T., Bridgett, S., Montazam, A., Nichols, J., Blaxter, M., & Gharbi, K. (2014). Quality control of next-generation sequencing data without a reference. Frontiers in Genetics, 5. https://doi.org/ARTN 11110.3389/fgene.2014.00111

28. S., A. (2010). FastQC: a quality control tool for high throughput sequence data. http://www.bioinformatics.babraham.ac.uk/projects/fastqc

29. He, B., Zhu, R., Yang, H., Lu, Q., Wang, W., Song, L., Sun, X., Zhang, G., Li, S., Yang, J., Tian, G., Bing, P., & Lang, J. (2020). Assessing the Impact of Data Preprocessing on Analyzing Next Generation Sequencing Data. Frontiers in bioengineering and biotechnology, 8, 817. https://doi.org/10.3389/fbioe.2020.00817

30. Bolger, A. M., Lohse, M., & Usadel, B. (2014). Trimmomatic: a flexible trimmer for Illumina sequence data. Bioinformatics, 30(15), 2114-2120. https://doi.org/10.1093/bioinformatics/btu170

31. Cox, M. P., Peterson, D. A., & Biggs, P. J. (2010). SolexaQA: At-a-glance quality assessment of Illumina second-generation sequencing data. BMC Bioinformatics, 11, 485. https://doi.org/10.1186/1471-2105-11-485

32. Li, H., & Durbin, R. (2009). Fast and accurate short read alignment with Burrows-Wheeler transform. Bioinformatics, 25(14), 1754-1760. https://doi.org/10.1093/bioinformatics/btp324

33. Li, H., & Durbin, R. (2010). Fast and accurate long-read alignment with Burrows-Wheeler transform. Bioinformatics, 26(5), 589-595. https://doi.org/10.1093/bioinformatics/btp698

34. Alser, M., Rotman, J., Deshpande, D. et al. Technology dictates algorithms: recent developments in read alignment. Genome Biol 22, 249 (2021). https://doi.org/10.1186/s13059-021-02443-7

35. Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., Marth, G., Abecasis, G., Durbin, R., & 1000 Genome Project Data Processing Subgroup (2009). The Sequence Alignment/Map format and SAMtools. Bioinformatics (Oxford, England), 25(16), 2078–2079.

36. Caboche, S., Audebert, C., Lemoine, Y., & Hot, D. (2014). Comparison of mapping algorithms used in high-throughput sequencing: application to Ion Torrent data. BMC Genomics, 15, 264. https://doi.org/10.1186/1471-2164-15-264

37. Smith, T. F., & Waterman, M. S. (1981). Identification of common molecular subsequences. Journal of molecular biology, 147(1), 195–197. https://doi.org/10.1016/0022-2836(81)90087-5

38. Van der Auwera, G. A., Carneiro, M. O., Hartl, C., Poplin, R., Del Angel, G., Levy-Moonshine, A., Jordan, T., Shakir, K., Roazen, D., Thibault, J., Banks, E., Garimella, K. V., Altshuler, D., Gabriel, S., & DePristo, M. A. (2013). From FastQ data to high confidence variant calls: the Genome Analysis Toolkit best practices pipeline. Curr Protoc Bioinformatics, 43(1110), 11 10 11-11 10 33. https://doi.org/10.1002/0471250953.bi1110s43

39. Danecek, P., Auton, A., Abecasis, G., Albers, C. A., Banks, E., DePristo, M. A., Handsaker, R. E., Lunter, G., Marth, G. T., Sherry, S. T., McVean, G., Durbin, R., & 1000 Genomes Project Analysis Group (2011). The variant call format and VCFtools. Bioinformatics (Oxford, England), 27(15), 2156–2158. https://doi.org/10.1093/bioinformatics/btr330

40. Nielsen, R., Paul, J. S., Albrechtsen, A. & Song, Y. S. Genotype and SNP calling from
next-generation sequencing data. Nat. Rev. Genet. 12, 443–451 (2011)

41. Poplin, R., Chang, PC., Alexander, D. et al. A universal SNP and small-indel variant caller using deep neural networks. Nat Biotechnol 36, 983–987 (2018). https://doi.org/10.1038/nbt.4235

42. Munoz-Barrera, A., Rubio-Rodriguez, L. A., Diaz-de Usera, A., Jaspez, D., Lorenzo-Salazar, J. M., Gonzalez-Montelongo, R., Garcia-Olivares, V., & Flores, C. (2022). From Samples to Germline and Somatic Sequence Variation: A Focus on Next-Generation Sequencing in Melanoma Research. Life (Basel), 12(11). https://doi.org/10.3390/life12111939

43. Hwang, S., Kim, E., Lee, I., & Marcotte, E. M. (2015). Systematic comparison of variant calling pipelines using gold standard personal exome variants. Sci Rep, 5, 17875. https://doi.org/10.1038/srep17875

44. Hansen, N. F. (2016). Variant Calling From Next Generation Sequence Data. Methods Mol Biol, 1418, 209-224. https://doi.org/10.1007/978-1-4939-3578-9_11

45. Hebbar, P., & Sowmya, S. K. (2022). Genomic Variant Annotation: A Comprehensive Review of Tools and Techniques. Intelligent Systems Design and Applications, Isda 2021, 418, 1057-1067. https://doi.org/10.1007/978-3-030-96308-8_98

46. arXiv:1207.3907

47. McKenna, A., Hanna, M., Banks, E., Sivachenko, A., Cibulskis, K., Kernytsky, A., Garimella, K., Altshuler, D., Gabriel, S., Daly, M., & DePristo, M. A. (2010). The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data. Genome Res, 20(9), 1297-1303. https://doi.org/10.1101/gr.107524.110

48. Larson, D. E., Harris, C. C., Chen, K., Koboldt, D. C., Abbott, T. E., Dooling, D. J., Ley, T. J., Mardis, E. R., Wilson, R. K., & Ding, L. (2012). SomaticSniper: identification of somatic point mutations in whole genome sequencing data. Bioinformatics (Oxford, England), 28(3), 311–317. https://doi.org/10.1093/bioinformatics/btr665

49. Kim, S., Scheffler, K., Halpern, A.L. et al. Strelka2: fast and accurate calling of germline and somatic variants. Nat Methods 15, 591–594 (2018). https://doi.org/10.1038/s41592-018-0051-x

50. Koboldt, D. C., Zhang, Q., Larson, D. E., Shen, D., McLellan, M. D., Lin, L., Miller, C. A., Mardis, E. R., Ding, L., & Wilson, R. K. (2012). VarScan 2: somatic mutation and copy number alteration discovery in cancer by exome sequencing. Genome Res, 22(3), 568-576. https://doi.org/10.1101/gr.129684.111

51. Tingting Gong and others, Detection of somatic structural variants from short-read next-generation sequencing data, Briefings in Bioinformatics, Volume 22, Issue 3, May 2021, bbaa056, https://doi.org/10.1093/bib/bbaa056

52. Pirooznia, M., Goes, F. S., & Zandi, P. P. (2015). Whole-genome CNV analysis: advances in computational approaches. Frontiers in genetics, 6, 138. https://doi.org/10.3389/fgene.2015.00138

53. de Ligt, J., Boone, P. M., Pfundt, R., Vissers, L. E., Richmond, T., Geoghegan, J., O'Moore, K., de Leeuw, N., Shaw, C., Brunner, H. G., Lupski, J. R., Veltman, J. A., & Hehir-Kwa, J. Y. (2013). Detection of clinically relevant copy number variants with

whole-exome sequencing. Hum Mutat, 34(10), 1439-1448.
https://doi.org/10.1002/humu.22387

54. Gabrielaite, M., Torp, M. H., Rasmussen, M. S., Andreu-Sanchez, S., Vieira, F. G., Pedersen, C. B., Kinalis, S., Madsen, M. B., Kodama, M., Demircan, G. S., Simonyan, A., Yde, C. W., Olsen, L. R., Marvig, R. L., Ostrup, O., Rossing, M., Nielsen, F. C., Winther, O., & Bagger, F. O. (2021). A Comparison of Tools for Copy-Number Variation Detection in Germline Whole Exome and Whole Genome Sequencing Data. Cancers (Basel), 13(24). https://doi.org/10.3390/cancers13246283

55. Nam, J. Y., Kim, N. K., Kim, S. C., Joung, J. G., Xi, R., Lee, S., Park, P. J., & Park, W. Y. (2016). Evaluation of somatic copy number estimation tools for whole-exome sequencing data. Brief Bioinform, 17(2), 185-192.
https://doi.org/10.1093/bib/bbv055

56. van Belzen, I., Schonhuth, A., Kemmeren, P., & Hehir-Kwa, J. Y. (2021). Structural variant detection in cancer genomes: computational challenges and perspectives for precision oncology. NPJ Precis Oncol, 5(1), 15.
https://doi.org/10.1038/s41698-021-00155-6

57. Klambauer, G., Schwarzbauer, K., Mayr, A., Clevert, D. A., Mitterecker, A., Bodenhofer, U., & Hochreiter, S. (2012). cn.MOPS: mixture of Poissons for discovering copy number variations in next-generation sequencing data with a low false discovery rate. Nucleic acids research, 40(9), e69. https://doi.org/10.1093/nar/gks003

58. Li, J., Lupat, R., Amarasinghe, K. C., Thompson, E. R., Doyle, M. A., Ryland, G. L., Tothill, R. W., Halgamuge, S. K., Campbell, I. G., & Gorringe, K. L. (2012). CONTRA: copy number analysis for targeted resequencing. Bioinformatics, 28(10), 1307-1313. https://doi.org/10.1093/bioinformatics/bts1467

59. Rausch, T. et al. DELLY: structural variant discovery by integrated paired-end and splitread analysis. Bioinformatics 28, i333–i339 (2012).

60. Plagnol, V., Curtis, J., Epstein, M., Mok, K. Y., Stebbings, E., Grigoriadou, S., Wood, N. W., Hambleton, S., Burns, S. O., Thrasher, A. J., Kumararatne, D., Doffinger, R., & Nejentsev, S. (2012). A robust model for read count data in exome sequencing experiments and implications for copy number variant calling. Bioinformatics, 28(21), 2747-2754. https://doi.org/10.1093/bioinformatics/bts526

61. https://gatk.broadinstitute.org/hc/en-us/articles/360035531092--How-to-part-I-Sensitively-detect-copy-ratio-alterations-and-allelic-segments

62. McCarthy, D. J. et al. Choice of transcripts and software has a large effect on variant
annotation. Genome Med. 6, 26 (2014).

63. Kircher, M., Witten, D., Jain, P. et al. A general framework for estimating the relative pathogenicity of human genetic variants. Nat Genet 46, 310–315 (2014). https://doi.org/10.1038/ng.2892

64. MacDonald, J. R., Ziman, R., Yuen, R. K., Feuk, L., & Scherer, S. W. (2014). The Database of Genomic Variants: a curated collection of structural variation in the human genome. Nucleic acids research, 42(Database issue), D986–D992. https://doi.org/10.1093/nar/gkt958

65. Frankish, A., Diekhans, M., Jungreis, I., Lagarde, J., Loveland, J. E., Mudge, J. M., Sisu, C., Wright, J. C., Armstrong, J., Barnes, I., Berry, A., Bignell, A., Boix, C., Carbonell Sala, S., Cunningham, F., Di Domenico, T., Donaldson, S., Fiddes, I. T., García Girón, C., Gonzalez, J. M., … Flicek, P. (2021). GENCODE 2021. Nucleic acids research, 49(D1), D916–D923. https://doi.org/10.1093/nar/gkaa1087

66. Chen S, Francioli LC, Goodrich JK, et al. A genome-wide mutational constraint map quantified from variation in 76,156 human genomes. bioRxiv; 2022. DOI: 10.1101/2022.03.20.485034.

67. Joanna S. Amberger and others, OMIM.org: Online Mendelian Inheritance in Man (OMIM®), an online catalog of human genes and genetic disorders, Nucleic Acids Research, Volume 43, Issue D1, 28 January 2015, Pages D789–D798, https://doi.org/10.1093/nar/gku1205

68. Geoffroy, V., Herenger, Y., Kress, A., Stoetzel, C., Piton, A., Dollfus, H., & Muller, J. (2018). AnnotSV: an integrated tool for structural variations annotation. Bioinformatics, 34(20), 3572-3574. https://doi.org/10.1093/bioinformatics/bty304

69. O'Leary, N. A., Wright, M. W., Brister, J. R., Ciufo, S., Haddad, D., McVeigh, R., Rajput, B., Robbertse, B., Smith-White, B., Ako-Adjei, D., Astashyn, A., Badretdin, A., Bao, Y., Blinkova, O., Brover, V., Chetvernin, V., Choi, J., Cox, E., Ermolaeva, O., Farrell, C. M., … Pruitt, K. D. (2016). Reference sequence (RefSeq) database at NCBI: current status, taxonomic expansion, and functional annotation. Nucleic acids research, 44(D1), D733–D745. https://doi.org/10.1093/nar/gkv1189

70. Landrum, M. J., Chitipiralla, S., Brown, G. R., Chen, C., Gu, B., Hart, J., Hoffman, D., Jang, W., Kaur, K., Liu, C., Lyoshin, V., Maddipatla, Z., Maiti, R., Mitchell, J., O'Leary, N., Riley, G. R., Shi, W., Zhou, G., Schneider, V., Maglott, D., … Kattman, B. L. (2020). ClinVar: improvements to accessing data. Nucleic acids research, 48(D1), D835–D844. https://doi.org/10.1093/nar/gkz972

71. Rehm, H. L., Berg, J. S., Brooks, L. D., Bustamante, C. D., Evans, J. P., Landrum, M. J., Ledbetter, D. H., Maglott, D. R., Martin, C. L., Nussbaum, R. L., Plon, S. E., Ramos, E. M., Sherry, S. T., Watson, M. S., & ClinGen (2015). ClinGen--the

Clinical Genome Resource. The New England journal of medicine, 372(23), 2235–2242. https://doi.org/10.1056/NEJMsr1406261
72. Riggs, E. R., Andersen, E. F., Cherry, A. M., Kantarci, S., Kearney, H., Patel, A., Raca, G., Ritter, D. I., South, S. T., Thorland, E. C., Pineda-Alvarez, D., Aradhya, S., & Martin, C. L. (2020). Technical standards for the interpretation and reporting of constitutional copy-number variants: a joint consensus recommendation of the American College of Medical Genetics and Genomics (ACMG) and the Clinical Genome Resource (ClinGen). Genetics in medicine : official journal of the American College of Medical Genetics, 22(2), 245–257. https://doi.org/10.1038/s41436-019-0686-8
73. McLaren, W., Gil, L., Hunt, S.E. et al. The Ensembl Variant Effect Predictor. Genome Biol 17, 122 (2016). https://doi.org/10.1186/s13059-016-0974-4
74. Vaser, R., Adusumalli, S., Leng, S. et al. SIFT missense predictions for genomes. Nat Protoc 11, 1–9 (2016). https://doi.org/10.1038/nprot.2015.123
75. Adzhubei, I., Schmidt, S., Peshkin, L. et al. A method and server for predicting damaging missense mutations. Nat Methods 7, 248–249 (2010). https://doi.org/10.1038/nmeth0410-248
76. Jalali Sefid Dashti, M., & Gamieldien, J. (2017). A practical guide to filtering and prioritizing genetic variants. BioTechniques, 62(1), 18–30. https://doi.org/10.2144/000114492
77. Guan, P., & Sung, W. K. (2016). Structural variation detection using next-generation sequencing data: A comparative technical review. Methods (San Diego, Calif.), 102, 36–49. https://doi.org/10.1016/j.ymeth.2016.01.020
78. Thorvaldsdóttir, H., Robinson, J. T., & Mesirov, J. P. (2013). Integrative Genomics Viewer (IGV): high-performance genomics data visualization and exploration. Briefings in bioinformatics, 14(2), 178–192. https://doi.org/10.1093/bib/bbs017
79. Brian T Lee and others, The UCSC Genome Browser database: 2022 update, Nucleic Acids Research, Volume 50, Issue D1, 7 January 2022, Pages D1115–D1122, https://doi.org/10.1093/nar/gkab959
80. Yokoyama, T.T., Kasahara, M. Visualization tools for human structural variations identified by whole-genome sequencing. J Hum Genet 65, 49–60 (2020). https://doi.org/10.1038/s10038-019-0687-0
81. De Cario, R., Kura, A., Suraci, S., Magi, A., Volta, A., Marcucci, R., Gori, A. M., Pepe, G., Giusti, B., & Sticchi, E. (2020). Sanger Validation of High-Throughput

Sequencing in Genetic Diagnosis: Still the Best Practice?. Frontiers in genetics, 11, 592588. https://doi.org/10.3389/fgene.2020.592588

82. Zhao, Y., Fang, L. T., Shen, T. W., Choudhari, S., Talsania, K., Chen, X., Shetty, J., Kriga, Y., Tran, B., Zhu, B., Chen, Z., Chen, W., Wang, C., Jaeger, E., Meerzaman, D., Lu, C., Idler, K., Ren, L., Zheng, Y., . . . Xiao, W. (2021). Whole genome and exome sequencing reference datasets from a multi-center and cross-platform benchmark study. Sci Data, 8(1), 296. https://doi.org/10.1038/s41597-021-01077-5

83. Mercer, T. R., Xu, J., Mason, C. E., Tong, W., & Consortium, M. S. (2021). The Sequencing Quality Control 2 study: establishing community standards for sequencing in precision medicine. Genome Biol, 22(1), 306. https://doi.org/10.1186/s13059-021-02528-3

84. Talsania, K., Shen, Tw., Chen, X. et al. Structural variant analysis of a cancer reference cell line sample using multiple sequencing technologies. Genome Biol 23, 255 (2022). https://doi.org/10.1186/s13059-022-02816-6

85. Kircher, M., Heyn, P., & Kelso, J. (2011). Addressing challenges in the production and analysis of illumina sequencing data. BMC Genomics, 12, 382. https://doi.org/10.1186/1471-2164-12-382

86. Danecek, P., Bonfield, J. K., Liddle, J., Marshall, J., Ohan, V., Pollard, M. O., Whitwham, A., Keane, T., McCarthy, S. A., Davies, R. M., & Li, H. (2021). Twelve years of SAMtools and BCFtools. Gigascience, 10(2). https://doi.org/10.1093/gigascience/giab008

87. You, N., Murillo, G., Su, X., Zeng, X., Xu, J., Ning, K., Zhang, S., Zhu, J., & Cui, X. (2012). SNP calling using genotype model selection on high-throughput sequencing data. Bioinformatics, 28(5), 643-650. https://doi.org/10.1093/bioinformatics/bts001

88. Lahens, N. F., Ricciotti, E., Smirnova, O., Toorens, E., Kim, E. J., Baruzzo, G., Hayer, K. E., Ganguly, T., Schug, J., & Grant, G. R. (2017). A comparison of Illumina and Ion Torrent sequencing platforms in the context of differential gene expression. BMC Genomics, 18(1), 602. https://doi.org/10.1186/s12864-017-4011-0

89. Polderman, T. J., Benyamin, B., de Leeuw, C. A., Sullivan, P. F., van Bochoven, A., Visscher, P. M., & Posthuma, D. (2015). Meta-analysis of the heritability of human traits based on fifty years of twin studies. Nat Genet, 47(7), 702-709. https://doi.org/10.1038/ng.3285

90. Zanti, M., Michailidou, K., Loizidou, M. A., Machattou, C., Pirpa, P., Christodoulou, K., Spyrou, G. M., Kyriacou, K., & Hadjisavvas, A. (2021). Performance evaluation of pipelines for mapping, variant calling and interval padding, for the

analysis of NGS germline panels. BMC bioinformatics, 22(1), 218.
https://doi.org/10.1186/s12859-021-04144-1

Attachments

A.1. Variant callers

A.1.1 . Number of chromosomes called per each Illumina variant caller

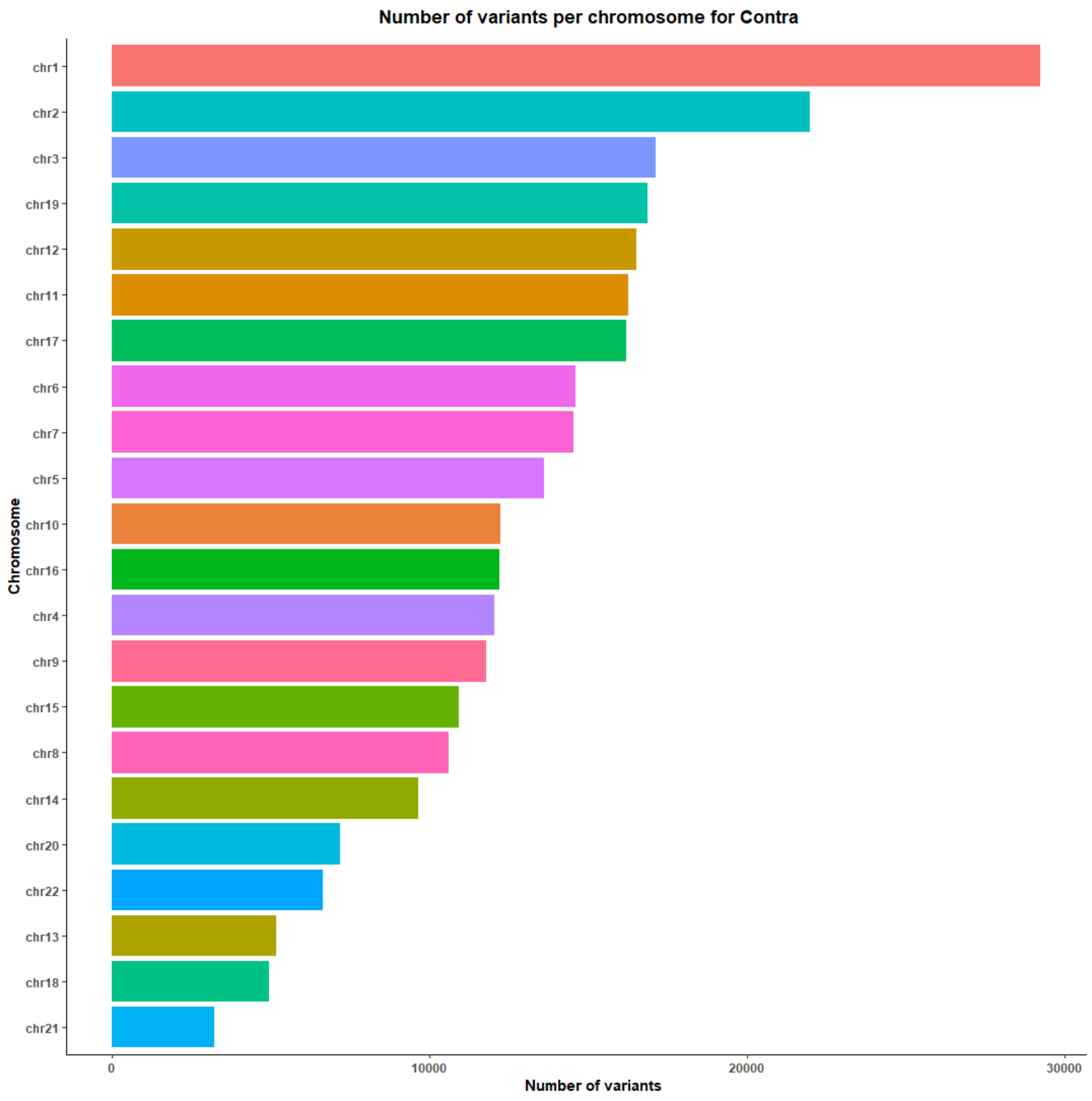


Figure 21 - Number of variants per chromosome for Contra

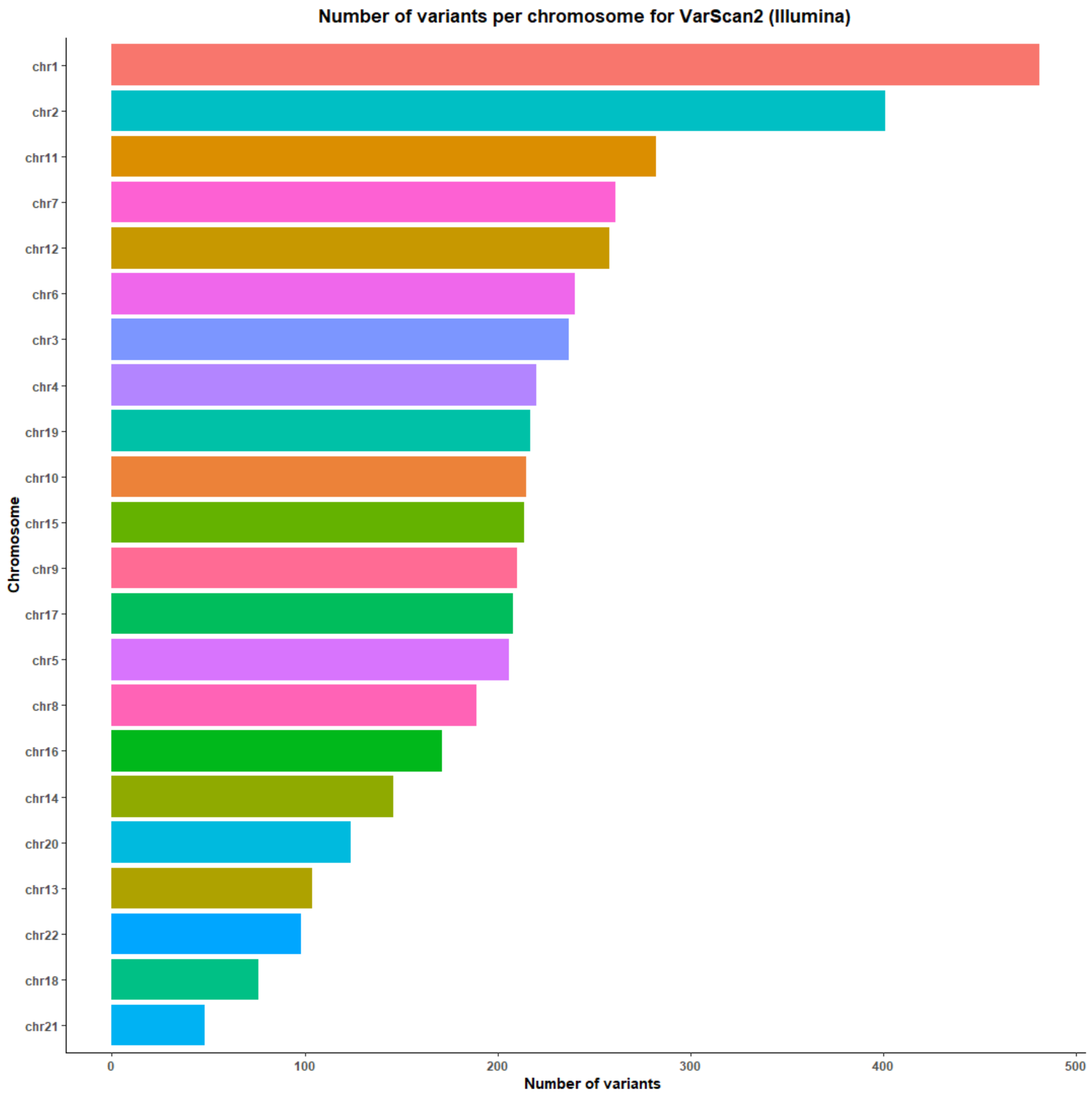


Figure 22 - Number of variants per chromosome for VarScan2 (Illumina)

A.1.1 . Number of chromosomes called per each Ion Torrent variant caller

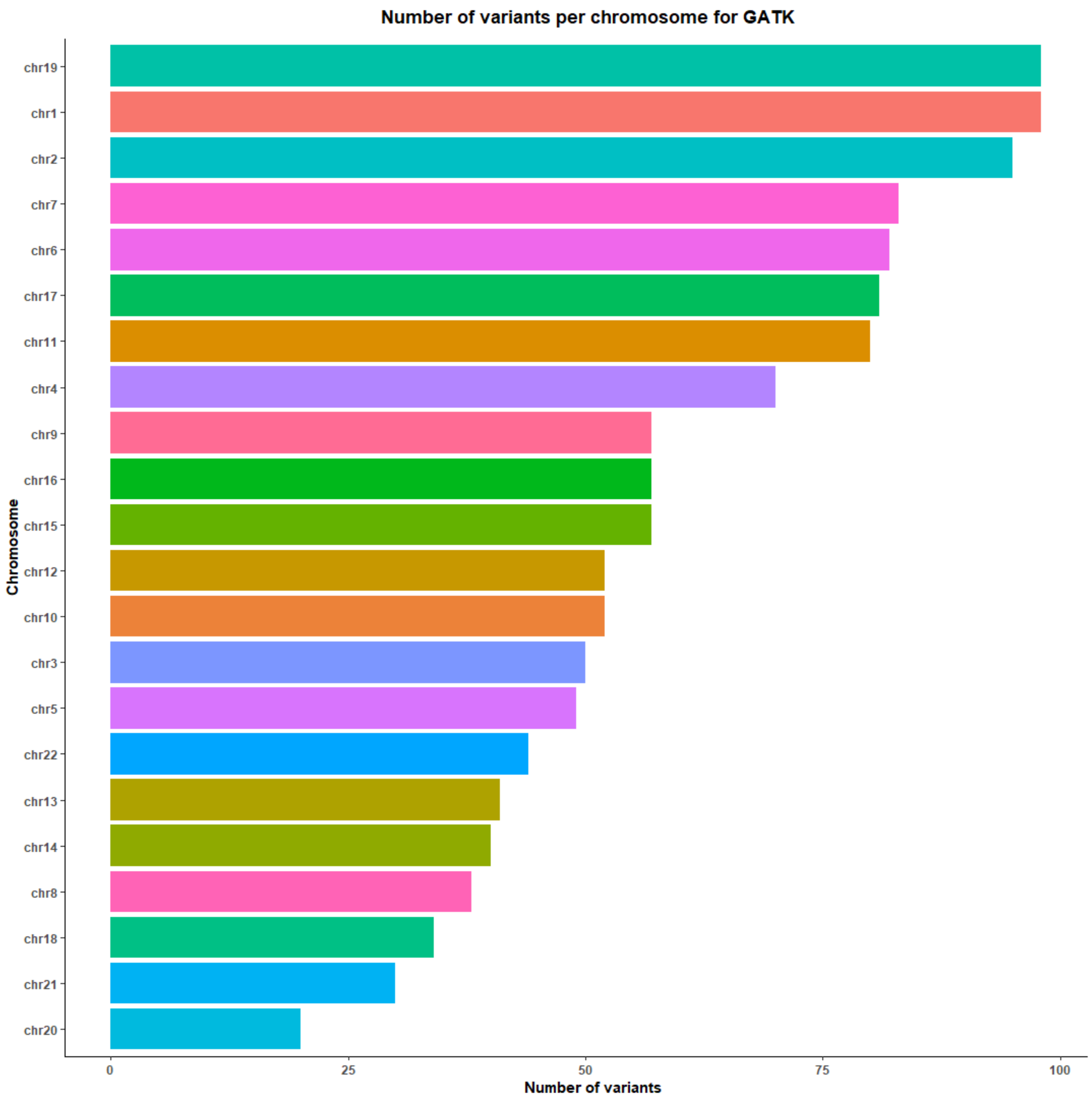


Figure 23 - Number of variants per chromosome for GATK

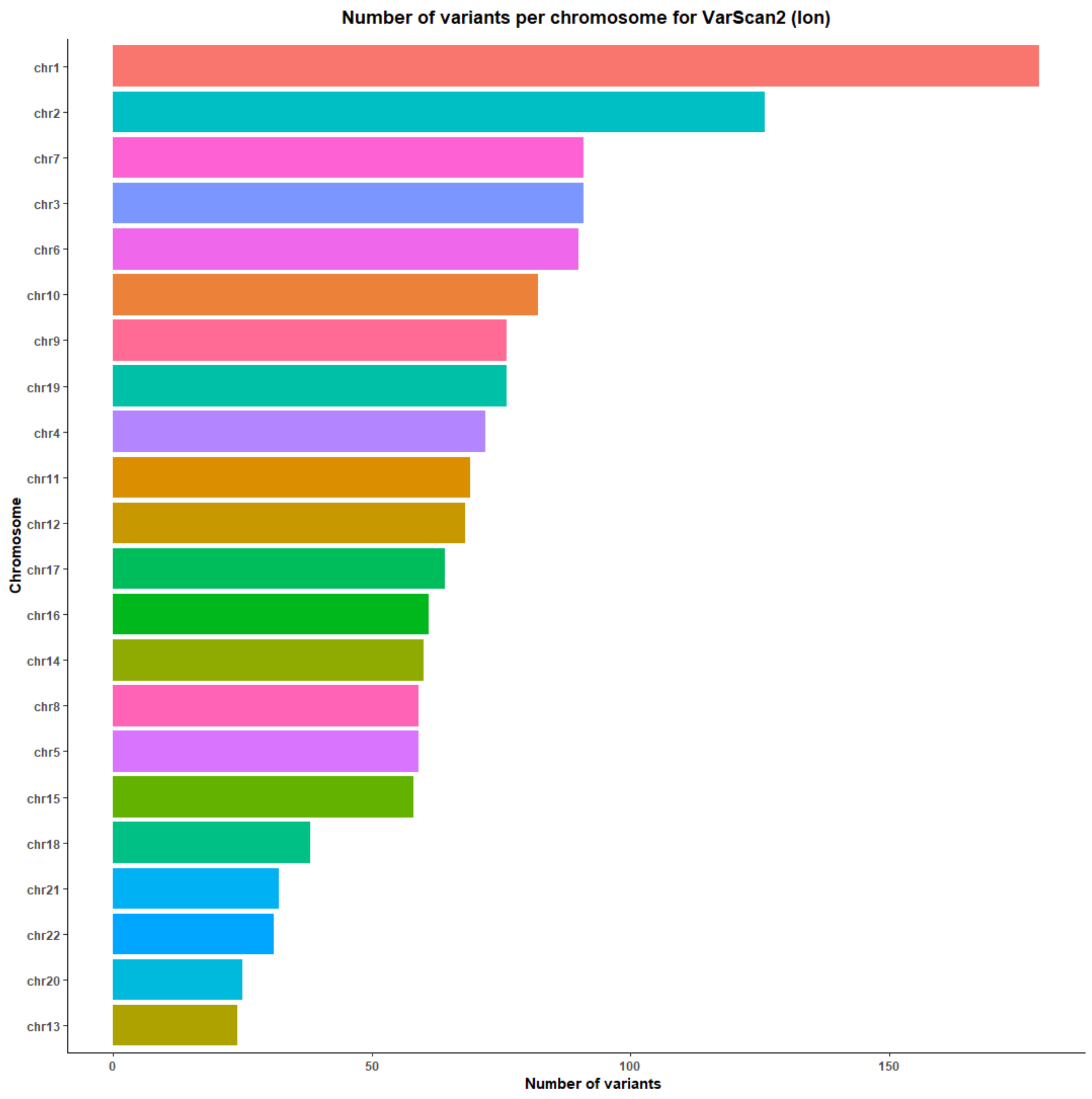


Figure 24 - Number of variants per chromosome for VarScan2 (Ion)

A.2. Pipeline

The script developed for this pipeline includes many functions that will be shown below. Our script assumes that people have already installed the necessary programs and dependencies.

```
#Libraries
import os
import sys
import re
import gzip
import shutil
import pandas as pd
from rpy2 import robjects



#Inputs

#Function to obtain the input files
def inpt(x):
    if x == "single":
        readn = input("What is the normal read? ")
        readt = input("What is the tumour read? ")
        sequencer = input("What was the sequencer used? ").lower()
        ref_genome = input("What is the genome of reference? ")
        index = input("Do you want to index the genome of reference? ").lower()
        ks = input("What is the file with the known polymorphic sites? ")
        bf = input("What is the target bed file? ")
        return x, readn, readt, sequencer, ref_genome, index, ks, bf
    elif x == "paired":
        read1n = input("What is the first normal read? ")
        read2n = input("What is the second normal read? ")
        read1t = input("What is the first tumour read? ")
        read2t = input("What is the second tumour read? ")
        sequencer = input("What was the sequencer used? ").lower()
        ref_genome = input("What is the genome of reference? ")
        index = input("Do you want to index the genome of reference? ").lower()
        ks = input("What is the file with the known polymorphic sites? ")
        bf = input("What is the target bed file? ")
        return x, read1n, read2n, read1t, read2t, sequencer, ref_genome, index, ks,
bf
```

```
#Header

#Function to obtain the parameters that will make part of the header
def parm_f(read):
    parm = []
    with open(read, "r") as f:
        for line in f:
            if line.startswith("@"):
                y = line.split(" ")
                for i in y:
                    z = i.split(":")
                    parm.extend(z)
                      return parm

#Function to obtain the header
def header(read, sequencer):
    r_name = re.sub(r"\..*", "", read)
    re_name = r_name.split("_")
    read_name = re_name[0]
    p_r = parm_f(read)
    if p_r[-1].endswith("\n"):
        p_r[-1] = p_r[-1].replace("\n","")
    p_r[0] = p_r[0].replace("@", "")
    ID = f"{read_name}"
    PL = f"{sequencer}"
    PU = f"{re_name[1]}"
    h = f"@RG\tID:{ID}\tPL:{PL}\tPU:{PU}\tSM:{ID}"
    return h




#Reads processing

#Function to obtain the processed reads in case they are single end reads
def read_processing_s(rn, rt, s):
    rn_name1 = re.split("\_", rn)
    rn_name2 = re.split("\.", rn_name1[2])
    rn_name = f"{rn_name1[0]}{rn_name1[1]}{rn_name2[1]}"
    rt_name1 = re.split("\_", rt)
    rt_name2 = re.split("\.", rt_name1[2])
    rt_name = f"{rt_name1[0]}{rt_name1[1]}{rt_name2[1]}"
    if s == "illumina":
        os.system(f"trimmomatic SE -phred33 {rn} {rn_name}_trimts.fastq
ILLUMINACLIP:/opt/Trimmomatic-0.39/adapters/TruSeq3-PE-2.fa:2:30:10 MINLEN:50")
        os.system(f"trimmomatic SE -phred33 {rt} {rt_name}_trimts.fastq
ILLUMINACLIP:/opt/Trimmomatic-0.39/adapters/TruSeq3-PE-2.fa:2:30:10 MINLEN:50")
    elif s == "ion torrent":
        os.system(f"SolexaQA++ dynamictrim {rn} --torrent")
        os.system(f"SolexaQA++ dynamictrim {rt} --torrent")
        os.system(f"SolexaQA++ lengthsort {rn_name}.fastq.trimmed -l 50")
        os.system(f"SolexaQA++ lengthsort {rt_name}.fastq.trimmed -l 50")
```

```
#Function to obtain the processed reads in case they are paired-end reads
def read_processing_p(r1n, r1t, r2n, r2t, s):
    r1n_name1 = re.split("\_", r1n)
    r1n_name2 = re.split("\.", r1n_name1[2])
    r1n_name = f"{r1n_name1[0]}{r1n_name1[1]}_{r1n_name2[1]}"
    r2n_name1 = re.split("\_", r2n)
    r2n_name2 = re.split("\.", r2n_name1[2])
    r2n_name = f"{r2n_name1[0]}{r2n_name1[1]}_{r2n_name2[1]}"
    r1t_name1 = re.split("\_", r1t)
    r1t_name2 = re.split("\.", r1t_name1[2])
    r1t_name = f"{r1t_name1[0]}{r1t_name1[1]}_{r1t_name2[1]}"
    r2t_name1 = re.split("\_", r2t)
    r2t_name2 = re.split("\.", r2t_name1[2])
    r2t_name = f"{r2t_name1[0]}{r2t_name1[1]}_{r2t_name2[1]}"
    if s == "illumina":
        os.system(f"trimmomatic PE -phred33 -trimlog {r1n_name}_trimlog {r1n} {r2n}
{r1n_name}_trimts.fastq {r1n_name}_trimtns.fastq {r2n_name}_trimts.fastq
{r2n_name}_trimtns.fastq ILLUMINACLIP:/opt/Trimmomatic-0.39/adapters/TruSeq3-PE-
2.fa:2:30:10 MINLEN:50")
        os.system(f"trimmomatic PE -phred33 -trimlog {r1t_name}_trimlog {r1t} {r2t}
{r1t_name}_trimts.fastq {r1t_name}_trimtns.fastq {r2t_name}_trimts.fastq
{r2t_name}_trimtns.fastq ILLUMINACLIP:/opt/Trimmomatic-0.39/adapters/TruSeq3-PE-
2.fa:2:30:10 MINLEN:50")
    elif s == "ion torrent":
        os.system(f"SolexaQA++ dynamictrim {r1n} {r2n} --torrent")
        os.system(f"SolexaQA++ dynamictrim {r1t} {r2t} --torrent")
        os.system(f"SolexaQA++ lengthsort {r1n_name}.fastq.trimmed
{r2n_name}.fastq.trimmed -l 50")
        os.system(f"SolexaQA++ lengthsort {r1t_name}.fastq.trimmed
{r2t_name}.fastq.trimmed -l 50")



#Alignment

#Function to align the processed reads in case they are single-end reads
def map_align_s(rn, rt, s, rg, i):
    rn_name = re.split(r"\..*", "", rn)
    rt_name = re.split(r"\..*", "", rt)
    rg_name = re.sub(r"\..*", "", rg)
    h_rn = header(rn, s)
    h_rn_parm = h_rn.split("\t")
    h_rt = header(rt, s)
    h_rt_parm = h_rt.split("\t")
    if s == "illumina":
        if i == "yes":
            os.system(f"samtools faidx {rg} chr1 chr2 chr3 chr4 chr5 chr6 chr7 chr8
chr9 chr10 chr11 chr12 chr13 chr14 chr15 chr16 chr17 chr18 chr19 chr20 chr21 chr22 >
{rg_name}.fa")
            os.system(f"bwa index -a bwtsw {rg_name}.fa")
            os.system(f"bwa mem -R '{h_rn}' {rg} {rn_name}_trimts.fastq | samtools
sort -o aln_{rn_name}.bam")
            os.system(f"bwa mem -R '{h_rt}' {rg} {rt_name}_trimts.fastq | samtools
sort -o aln_{rt_name}.bam")
            os.system(f"samtools index aln_{rn_name}bam")
            os.system(f"samtools index aln_{rt_name}bam")
        elif i == "no":
            os.system(f"bwa mem -R '{h_rn}' {rg} {rn_name}_trimts.fastq | samtools
sort -o aln_{rn_name}.bam")
            os.system(f"bwa mem -R '{h_rt}' {rg} {rt_name}_trimts.fastq | samtools
sort -o aln_{rt_name}.bam")
            os.system(f"samtools index aln_{rn_name}bam")
            os.system(f"samtools index aln_{rt_name}bam")
    elif s == "ion torrent":
        if i == "yes":
            os.system(f"tmap-ion index -f {rg_name}.fa")
            os.system(f"tmap-ion mapall -f {rg} -r {rn}.trimmed -v -Y -u -R
{h_rn_parm[1]},{h_rn_parm[2]} ,{h_rn_parm[3]},{h_rn_parm[4]} stage1 map1 map2 map3
map4 | samtools sort -o aln_{rn_name}.bam")
            os.system(f"tmap-ion mapall -f {rg} -r {rt}.trimmed -v -Y -u -R
{h_rt_parm[1]},{h_rt_parm[2]},{h_rt_parm[3]},{h_rt_parm[4]} stage1 map1 map2 map3
map4 | samtools sort -o aln_{rt_name}.bam")
        elif i == "no":
            os.system(f"tmap-ion mapall -f {rg} -r {rn}.trimmed -v -Y -u -R
{h_rn_parm[1]},{h_rn_parm[2]},{h_rn_parm[3]},{h_rn_parm[4]} stage1 map1 map2 map3
map4 | samtools sort -o aln_{rn_name}.bam")
            os.system(f"tmap-ion mapall -f {rg} -r {rt}.trimmed -v -Y -u -R
{h_rt_parm[1]},{h_rt_parm[2]},{h_rt_parm[3]},{h_rt_parm[4]} stage1 map1 map2 map3
map4 | samtools sort -o aln_{rt_name}.bam")


#Function to align the processed reads in case they are paired end reads
def map_align_p(r1n, r2n, r1t, r2t, s, rg, i, ks):
    r1n_name1 = re.split("\_", r1n)
    r1n_name2 = re.split("\.", r1n_name1[2])
    r1n_name = f"{r1n_name1[0]}{r1n_name1[1]}_{r1n_name2[1]}"
    r2n_name1 = re.split("\_", r2n)
    r2n_name2 = re.split("\.", r2n_name1[2])
    r2n_name = f"{r2n_name1[0]}{r2n_name1[1]}_{r2n_name2[1]}"
    r1t_name1 = re.split("\_", r1t)
    r1t_name2 = re.split("\.", r1t_name1[2])
    r1t_name = f"{r1t_name1[0]}{r1t_name1[1]}_{r1t_name2[1]}"
    r2t_name1 = re.split("\_", r2t)
    r2t_name2 = re.split("\.", r2t_name1[2])
    r2t_name = f"{r2t_name1[0]}{r2t_name1[1]}_{r2t_name2[1]}"
    rg_name = re.sub(r"\..*", "", rg)
    h_rn = header(r1n, s)
    h_rn_parm = h_rn.split("\t")
    h_rt = header(r1t, s)
    h_rt_parm = h_rt.split("\t")
    if s == "illumina":
        if i == "yes":
            os.system(f"samtools faidx {rg} chr1 chr2 chr3 chr4 chr5 chr6 chr7 chr8
chr9 chr10 chr11 chr12 chr13 chr14 chr15 chr16 chr17 chr18 chr19 chr20 chr21 chr22 >
{rg_name}.fa")
            os.system(f"bwa index {rg_name}.fa")
            os.system(f"bwa mem -R '{h_rn}' {rg_name}.fa {r1n_name}_trimts.fastq
{r2n_name}_trimts.fastq | samtools sort -o aln_{r1n_name[0]}.bam")
            os.system(f"bwa mem -R '{h_rt}' {rg_name}.fa {r1t_name}_trimts.fastq
{r2t_name}_trimts.fastq | samtools sort -o aln_{r1t_name[0]}.bam")
            os.system(f"samtools index aln_{r1n_name[0]}.bam")
            os.system(f"samtools index aln_{r1t_name[0]}.bam")
        elif i == "no":
            os.system(f"bwa mem -R '{h_rn}' {rg_name}.fa {r1n_name}_trimts.fastq
{r2n_name}_trimts.fastq | samtools sort -o aln_{r1n_name[0]}.bam")
            os.system(f"bwa mem -R '{h_rt}' {rg_name}.fa {r1t_name}_trimts.fastq
{r2t_name}_trimts.fastq | samtools sort -o aln_{r1t_name[0]}.bam")
            os.system(f"samtools index aln_{r1n_name[0]}.bam")
            os.system(f"samtools index aln_{r1t_name[0]}.bam")
    elif s == "ion torrent":
        if i == "yes":
            os.system(f"tmap-ion index -f {rg_name}.fa")
            os.system(f"tmap-ion mapall -f {rg_name}.fa -r {r1n} {r2n} -S 1 -P 0 -v -
u -R {h_rn_parm[1]} -R {h_rn_parm[2]} -R {h_rn_parm[3]} -R {h_rn_parm[4]} stage1 map1
map2 map3 map4 | samtools sort -o aln_{r1n_name[0]}.bam")
            os.system(f"tmap-ion mapall -f {rg_name}.fa -r {r1t} {r2t} -S 1 -P 0 -v -
u -R {h_rt_parm[1]} -R {h_rt_parm[2]} -R {h_rt_parm[3]} -R {h_rt_parm[4]} stage1 map1
map2 map3 map4 | samtools sort -o aln_{r1t_name[0]}.bam")
        elif i == "no":
            os.system(f"tmap-ion mapall -f {rg_name}.fa -r {r1n} {r2n} -S 1 -P 0 -v -
u -R {h_rn_parm[1]} -R {h_rn_parm[2]} -R {h_rn_parm[3]} -R {h_rn_parm[4]} stage1 map1
map2 map3 map4 | samtools sort -o aln_{r1n_name[0]}.bam")
            os.system(f"tmap-ion mapall -f {rg_name}.fa -r {r1t} {r2t} -S 1 -P 0 -v -
u -R {h_rt_parm[1]} -R {h_rt_parm[2]} -R {h_rt_parm[3]} -R {h_rt_parm[4]} stage1 map1
map2 map3 map4 | samtools sort -o aln_{r1t_name[0]}.bam")




#BAMs processing

#Marking Duplicates and Base Quality Score Recalibration
def markdup_bqsr(r1n, r1t, rg, ks):
    r1n_name1 = re.split("\_", r1n)
    r1n_name2 = re.split("\.", r1n_name1[2])
    r1n_name = f"{r1n_name1[0]}{r1n_name1[1]}_{r1n_name2[1]}"
    r1t_name1 = re.split("\_", r1t)
    r1t_name2 = re.split("\.", r1t_name1[2])
    r1t_name = f"{r1t_name1[0]}{r1t_name1[1]}_{r1t_name2[1]}"
    rg_name = re.sub(r"\..*", "", rg)
    os.system("awk 'FNR > 62 {print $1, $10}' <
GCF_000001405.38_GRCh38.p12_assembly_report.txt | awk 'FNR > 1 {print}' >
rename_chromosomes_map.txt")
    os.system(f"bcftools annotate --rename-chrs rename_chromosomes_map.txt {ks} >
All_new.vcf")
    os.system("gatk IndexFeatureFile -I All_new.vcf")
    os.system(f"samtools faidx {rg_name}.fa")
    os.system(f"gatk CreateSequenceDictionary R={rg_name}.fa O={rg_name}.dict")
    os.system(f"gatk MarkDuplicates I=aln_{r1n_name[0]}.bam
O=dup_aln_{r1n_name[0]}.bam M=dup_metrics_{r1n_name[0]}.txt ASSUME_SORTED=TRUE
VALIDATION_STRINGENCY=LENIENT")
    os.system(f"gatk MarkDuplicates I=aln_{r1n_name[0]}.bam
O=dup_aln_{r1t_name[0]}.bam M=dup_metrics_{r1t_name[0]}.txt ASSUME_SORTED=TRUE
VALIDATION_STRINGENCY=LENIENT")
    os.system(f"gatk BaseRecalibrator -R {rg_name}.fa -I aln_{r1n_name[0]}_m.bam --
known-sites {ks} -O rec_aln_{r1n_name[0]}.table")
    os.system(f"gatk BaseRecalibrator -R {rg_name}.fa -I aln_{r1t_name[0]}_m.bam --
known-sites {ks} -O rec_aln_{r1t_name[0]}_.table")
    os.system(f"gatk BaseRecalibrator -R {rg_name}.fa -I aln_{r1n_name[0]}_m.bam --
known-sites {ks} -O rec_sec_aln_{r1n_name[0]}.table")
    os.system(f"gatk BaseRecalibrator -R {rg_name}.fa -I aln_{r1t_name[0]}_m.bam --
known-sites {ks} -O rec_sec_aln_{r1t_name[0]}.table")
    os.system(f"gatk ApplyBQSR -R {rg_name}.fa -I dup_aln_{r1n_name[0]}.bam  --bqsr-
recal-file rec_sec_aln_{r1n_name[0]}.table -O rec_aln_{r1n_name[0]}.bam")
    os.system(f"gatk ApplyBQSR -R {rg_name}.fa -I dup_aln_{r1t_name[0]}.bam  --bqsr-
recal-file rec_sec_aln_{r1t_name[0]}.table -O rec_aln_{r1t_name[0]}.bam")




#Variant calling

def filter_target(bf):
    os.system(f"sort-bed {bf} > target.bed")
    os.system(f"awk "'{print $1, $2, $3}'" target.bed > target_reg.bed")
    for c in range(1,23):
        os.system(f"bedextract chr{c} target_reg.bed > target_{c}.bed")
    os.system(f"bedops -u target_1.bed target_2.bed target_3.bed target_4.bed
target_5.bed target_6.bed target_7.bed target_8.bed target_9.bed target_10.bed
target_11.bed target_12.bed target_13.bed target_14.bed target_15.bed target_16.bed
target_17.bed target_18.bed target_19.bed target_20.bed target_21.bed target_22.bed >
target_final.bed")
    os.system(f"sort -k1,1V target_final.bed > target_regions.bed")
```

```
#Illumina

#VarScan2
def var_call_vs2(r1n, r1t, rg):
    r1n_name1 = re.split("\_", r1n)
    r1n_name2 = re.split("\.", r1n_name1[2])
    r1n_name = f"{r1n_name1[0]}{r1n_name1[1]}_{r1n_name2[1]}"
    r1t_name1 = re.split("\_", r1t)
    r1t_name2 = re.split("\.", r1t_name1[2])
    r1t_name = f"{r1t_name1[0]}{r1t_name1[1]}_{r1t_name2[1]}"
    rg_name = re.sub(r"\..*", "", rg)
    os.system(f"samtools mpileup -q 1 -f {rg_name}.fa rec_aln_{r1n_name[0]}.bam 
rec_aln_{r1t_name[0]}.bam | java -jar /usr/local/bin/VarScan.v2.4.6.jar copynumber 
varScan --mpileup 1 --output-vcf 1")
    os.system("java -jar /usr/local/bin/VarScan.v2.4.6.jar copyCaller 
output.copynumber --output-file varScan.copynumber.called")
    robjects.r('''
        library(DNAcopy)
        cn <- read.table("varScan.copynumber.called",header=F)
        CNA.object <-CNA( genomdat = as.numeric(cn[,6]), chrom = cn[,1], maploc = 
as.numeric(cn[,2]), data.type = 'logratio')
        CNA.smoothed <- smooth.CNA(CNA.object)
        segs <- segment(CNA.smoothed, verbose=0, min.width=2)
        segs2 = segs$output
        write.table(segs2[,2:6], file="out.file", row.names=F, col.names=T, quote=F, 
sep="\t")
               ''')



#Contra
def var_call_contra(r1n,r1t,rg):
    r1n_name1 = re.split("\_", r1n)
    r1n_name2 = re.split("\.", r1n_name1[2])
    r1n_name = f"{r1n_name1[0]}{r1n_name1[1]}_{r1n_name2[1]}"
    r1t_name1 = re.split("\_", r1t)
    r1t_name2 = re.split("\.", r1t_name1[2])
    r1t_name = f"{r1t_name1[0]}{r1t_name1[1]}_{r1t_name2[1]}"
    rg_name = re.sub(r"\..*", "", rg)
    os.system("conda activate py27")
    os.system(f"contra.py --target target_regions.bed --test recaln_{r1t_name[0]}.bam 
--control rec_aln_{r1n_name[0]}.bam --fasta {rg_name}.fa -p -l --outFolder 
~/ContraTest/")




#Ion Torrent
def var_call_g(r1n, r1t, rg):
    r1n_name1 = re.split("\_", r1n)
    r1n_name2 = re.split("\.", r1n_name1[2])
    r1n_name = f"{r1n_name1[0]}{r1n_name1[1]}_{r1n_name2[1]}"
    r1t_name1 = re.split("\_", r1t)
    r1t_name2 = re.split("\.", r1t_name1[2])
    r1t_name = f"{r1t_name1[0]}{r1t_name1[1]}_{r1t_name2[1]}"
    rg_name = re.sub(r"\..*", "", rg)
    os.system(f"gsutil cp gs://broad-public-
datasets/funcotator/gnomAD_2.1_VCF_INFO_AF_Only/hg38/gnomad.exomes.r2.1.sites.liftove
rToHg38.INFO_ANNOTATIONS_FIXED.vcf.gz .")
    os.system(f"bcftools view -r 
chr1,chr2,chr3,chr4,chr5,chr6,chr7,chr8,chr9,chr10,chr11,chr12,chr13,chr14,chr15,chr1
6,chr17,chr18,chr19,chr20,chr21,chr22 
gnomAD_2.1_VCF_INFO_AF_Only/hg38/gnomad.exomes.r2.1.sites.liftoverToHg38.INFO_ANNOTAT
IONS_FIXED.vcf.gz > gnomad_exomes.vcf ")
```

```
    os.system(f"docker run -v
/home/imarques/pipeline/ion_torrent_samples:/gatk/my_data -it
broadinstitute/gatk:4.1.3.0")
    os.system(f"gatk PreprocessIntervals -L target_regions.bed -R {rg_name}.fa --bin-
length 0 --interval-merging-rule OVERLAPPING_ONLY -O targets.interval_list")
    os.system(f"gatk CollectReadCounts -I rec_aln_{r1n_name[0]}.bam -L
targets.interval_list --interval-merging-rule OVERLAPPING_ONLY -DF
ProperlyPairedReadFilter -O aln_{r1n_name[0]}.counts.hdf5")
    os.system(f"gatk CollectReadCounts -I rec_aln_{r1t_name[0]}.bam -L
targets.interval_list --interval-merging-rule OVERLAPPING_ONLY -DF
ProperlyPairedReadFilter -O aln_{r1t_name[0]}.counts.hdf5")
    os.system(f"gatk --java-options '-Xmx6500m' CreateReadCountPanelOfNormals -I
normal.counts.hdf5 --minimum-interval-median-percentile 0.5 -O cnvponC.pon.hdf5")
    os.system(f"gatk AnnotateIntervals -R {rg_name}.fa -L target_regions.bed --
interval-merging-rule OVERLAPPING_ONLY -O annotated_intervals.tsv")
    os.system(f"gatk --java-options '-Xmx12g' DenoiseReadCounts -I tumor.counts.hdf5
--count-panel-of-normals cnvponC.pon.hdf5 --standardized-copy-ratios
tumor.standardizedCR.tsv --denoised-copy-ratios tumor.denoisedCR.tsv --annotated-
intervals annotated_intervals.tsv")
    os.system(f"gatk PlotDenoisedCopyRatios --standardized-copy-ratios
tumor.standardizedCR.tsv --denoised-copy-ratios tumor.denoisedCR.tsv --sequence-
dictionary {rg_name}.dict --output ./plots --output-prefix tumor")
    os.system(f"gatk SelectVariants -R GRCh38.fa -V gnomad_exomes.vcf --select-type-
to-include SNP -O targets_SNP.vcf")
    os.system(f"gatk --java-options '-Xmx12g' CollectAllelicCounts -L gnomad_sv.vcf -
I recal_reads_t_aln81nm.bam -R GRCh38.fa -O normal.allelicCounts.tsv")
    os.system(f"gatk --java-options '-Xmx12g' CollectAllelicCounts -L gnomad_sv.vcf -
I recal_reads_t_aln82tm.bam -R GRCh38.fa -O tumor.allelicCounts.tsv")
    os.system(f"gatk --java-options '-Xmx8g' ModelSegments --denoised-copy-ratios
tumor.denoisedCR.tsv --allelic-counts tumor.allelicCounts.tsv --normal-allelic-counts
normal.allelicCounts.tsv --output ./segments --output-prefix tumor")
    os.system(f"gatk CallCopyRatioSegments ./segments/tumor.cr.seg --output
tumor.called.seg ")
    os.system(f"gatk PlotModeledSegments --denoised-copy-ratios tumor.denoisedCR.tsv
--allelic-counts ./segments/tumor.hets.tsv --segments ./segments/tumor.modelFinal.seg
--sequence-dictionary GRCh38_f.dict --minimum-contig-length 46709983 --output
./segments --output-prefix tumor")
    os.system(f"exit")



#Merging

#Illumina
def read_split_illumina(file, vc):
    current_file = pd.read_csv(file, sep="\t")
    if vc == "contra":
        new_file = current_file[["Chr","OriStCoordinate","OriEndCoordinate"]]
        new_file.insert(3, "caller", "contra")
        new_file.to_csv('contra.bed', sep = '\t', header = False, index = False)
    elif vc == "varscan":
        new_file = current_file[["chrom","loc.start","loc.end"]]
        new_file.insert(3, "caller", "varscan2")
        new_file.to_csv('varscan2_illumina.bed', sep = '\t', header = False, index =
False)


def mergefiles_illumina(file1, file2):
    read_split_ion(file1, "contra")
    read_split_ion(file2, "varscan")
    os.system("bedtools intersect -a contra.bed -b varscan2_illumina.bed -wao >
merge_illumina.bed")
```

```
def percentage_overlap_illumina(file):
    current_file = pd.read_csv(file, sep="\t")
    current_file.columns = ["chr", "chr_start_contra", "chr_end_contra", "caller",
"chr", "chr_start_var", "chr_end_var", "caller", "overlap"]
    size_contra = current_file["chr_end_contra"] - current_file["chr_start_contra"]
    size_varscan = current_file["chr_end_var"] - current_file["chr_start_var"]
    dif_contra = (current_file["overlap"] *100) / size_varscan
    dif_varscan = (current_file["overlap"] *100) / size_contra
    current_file['overlap_perc_contra'] = round(dif_contra,2)
    current_file['overlap_perc_varscan'] = round(dif_varscan,2)
    current_file.to_csv('merge_illumina.bed', sep = '\t', header = False, index =
False)
    return




#Ion Torrent
def read_split_ion(file, vc):
    current_file = pd.read_csv(file, sep="\t")
    if vc == "gatk":
        new_file = current_file[["CONTIG","START","END"]]
        new_file.insert(3, "caller", "gatk")
        new_file.to_csv('gatk.bed', sep = '\t', header = False, index = False)
    elif vc == "varscan":
        new_file = current_file[["chrom","loc.start","loc.end"]]
        new_file.insert(3, "caller", "varscan2")
        new_file.to_csv('varscan2_ion.bed', sep = '\t', header = False, index =
False)


def mergefiles_ion(file1, file2):
    os.system(f"sed '1,24d' {file1}")
    read_split_ion(file1, "gatk")
    read_split_ion(file2, "varscan")
    os.system("bedtools intersect -a gatk.bed -b varscan2_ion.bed -wao >
merge_ion.bed")


def percentage_overlap_ion(file):
    current_file = pd.read_csv(file, sep="\t")
    current_file.columns = ["chr", "chr_start_gatk", "chr_end_gatk", "caller", "chr",
"chr_start_var", "chr_end_var", "caller", "overlap"]
    size_gatk = current_file["chr_end_gatk"] - current_file["chr_start_gatk"]
    size_varscan = current_file["chr_end_var"] - current_file["chr_start_var"]
    dif_gatk = (current_file["overlap"] *100) / size_varscan
    dif_varscan = (current_file["overlap"] *100) / size_gatk
    current_file['overlap_perc_gatk'] = round(dif_gatk,2)
    current_file['overlap_perc_varscan'] = round(dif_varscan,2)
    current_file.to_csv('merge_ion.bed', sep = '\t', header = False, index = False)
    return



#Run all functions

x = input("Are the reads single or paired end? ").lower()

def calling(x):
  ipt = inpt(x)
  if ipt[0] == "single":
    file_merge = "merge_ion.bed"
    file1 = "tumor.called.seg"
    file2 = "out.file"
```

```
        read_processing_s(ipt[1], ipt[2], ipt[3])
        map_align_s(ipt[1], ipt[2], ipt[3], ipt[4], ipt[5], ipt[6])
        markdup_bqsr(ipt[1], ipt[2], ipt[4], ipt[6])
        filter_target(ipt[7])
        var_call_g(ipt[1], ipt[2], ipt[4])
        var_call_vs2(ipt[1], ipt[2], ipt[4])
        mergefiles_ion(file1,file2)
        percentage_overlap_ion(file_merge)
    elif ipt[0] == "paired":
        file_merge = "merge_illumina.bed"
        file1 = "CNATable.10rd.10bases.20bins.txt"
        file2 = "out.file"
        read_processing_p(ipt[1], ipt[2], ipt[3], ipt[4], ipt[5])
        map_align_p(ipt[1], ipt[2], ipt[3], ipt[4], ipt[5], ipt[6], ipt[7], ipt[8])
        markdup_bqsr(ipt[1], ipt[3], ipt[5], ipt[8])
        filter_target(ipt[9])
        var_call_contra(ipt[1], ipt[3], ipt[5])
        var_call_vs2(ipt[1], ipt[3], ipt[5])
        mergefiles_illumina(file1,file2)
        percentage_overlap_illumina(file_merge)


calling()
```