MSE | MASTER OF SCIENCE IN ENGINEERING

Hes·so
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences and Arts Western Switzerland

Master of Science HES-SO in Engineering
Av. de Provence 6
CH-1007 Lausanne

Master of Science HES-SO in Engineering

Orientation: Information and Communication Technologies (ICT)

Copy Number Variant Assessment of Whole-Genome Sequencing Data

Author:
Rick Wertenbroek

Under the direction of:
Prof. Yann Thoma
HEIG-VD - HES-SO
and
Prof. Ioannis Xenarios
UNIL - CHUV

Lausanne, HES-SO//Master, May 26, 2020

Information about this report

Contact information

Author: Rick Wertenbroek
MSE Student
HES-SO//Master
Switzerland
Email: *rick.wertenbroek@heig-vd.ch*

Declaration of honor

I, undersigned, Rick Wertenbroek, hereby declare that the work submitted is the result of a personal work. I certify that I have not resorted to plagiarism or other forms of fraud. All sources of information used and the author quotes were clearly mentioned.

Place, date: _____

Signature: _____

Acknowledgments

I would like to express my deep gratitude to Prof. Yann Thoma and Prof. Ioannis Xenarios for the constructive suggestions and support given throughout this project.

I would also like to thank the team at the Genome Center for their invaluable expertise and help provided with the sequencing data and related bioinformatic tools.

I would also like to thank all my friends and colleagues for their support and helpful comments.

Finally, I would like to thank Nathalie Mégevand for her support and encouragement throughout this project. As well as for her patience every time I went down the rabbit hole working late hours on this project.

Abstract

Copy number variation represents a substantial amount of the genetic diversity between individuals. It has been associated with disorders and gene expression but remains less studied and understood compared to smaller variations such as single nucleotide mutations. Recent studies of copy number variation have shown their diversity in size and type as well as their broad impact on phenotype and processes in humans and organisms in general. This work proposes tools for the discovery and analysis of copy number variations such as deletions, duplications, inversions, and putative insertions of DNA sequences in genomes based on next-generation sequencing data. The different kinds of variations are studied and methods are proposed for their discovery. Based on validation of these methods, tools have been created for automated discovery and classification. The performance of these tools are evaluated on simulated and real data sets. This results in a framework that can extract copy number variants as well as give predictions of regions that exhibit signs of structural variation on the whole genome.

Key words: Genomics, DNA Sequencing, Copy Number Variants (CNVs), Structural Variation (SV)

Résumé

La variation structurelle représente une partie conséquente de la diversité génomique. Ces variations dans le nombre de copies de chromosomes, gènes ou sous-séquences ADN ont été liées à des phénotypes ainsi qu'à des maladies génétiques mais restent bien moins étudiées que les petites variations telles que les mutations d'une seule base. Des études récentes ont montré la vaste diversité de ces variations, tant en taille qu'en nombre, ainsi que leur large spectre d'impact sur le phénotype, le fonctionnement de la cellule et sur l'organisme en général. Ce travail présente une collection d'outils développés pour détecter, quantifier et classifier ces variations. Les méthodes de détection sont présentées, suivies du développement d'outils de détection et de classification automatique pour finir sur l'évaluation des résultats sur des données de séquençage réelles et simulées. Le travail effectué résulte en une collection d'outils qui permettent d'investiguer les variations dans le génome entier depuis les données de séquençage.

Mots clés : Genomique, Séquençage ADN, Variation structurelle

Contents

List of Figures

List of Tables

1 Introduction

1.1 Context

This master project was done in collaboration with the health 2030 Genome Center `https://www.health2030genome.ch` in order to develop new tools for their genomic data analysis and interpretation platform.

1.2 Motivation

The motivation behind this project is to provide means to assess copy number variations (CNVs), a form of structural variation in individual genomes in order to provide new data to interpret when analyzing whole-genome sequencing data from patients or individuals for studies in genomics.

CNVs have been linked to phenotype, gene expression and regulation, 3D structure of proteins, as well as processes in the cell and the organism in general. They have a broad impact spectrum. However, the consequences of structural variation in the genome remain less studied than smaller mutations.

The possibility to efficiently extract CNV information from sequencing data will help improve the subsequent analyses. This will lead to better opportunities to study their broad impact and assess their roles in genetic diseases. With the more global motivation to further improve the understanding of the human genome, ameliorate human health conditions, and provide results that can be used for research in personalized medicine.

1.3 Objectives

The main objective of this project is to provide a tool to extract the breakpoints of copy number variant events, i.e., the locations, coordinates where the genome under study differs from the human reference by a form of copy number variation from next-generation whole-genome sequencing data.

The remaining objectives are, to analyze the different kinds of CNVs, provide strategies to detect and classify them, and finally benchmark the tool created and compare it to the state of the art.

1.4 Structure of this report

This report is structured as follows :

- **Background** This chapter covers the background of the project. It will serve as an introduction to the project and describe copy number variations.
- **Analysis** The analysis chapter covers the current state of the art of copy number variation detection as well as the existing biocomputing software and associated techniques.
- **Strategies and Algorithms** This chapter introduces strategies and algorithms that were used throughout this project.
- **Design** This chapter describes the design approaches taken to create copy number variant assessment tools as well as a more general framework to explore and benchmark the created tools.
- **Implementation** This chapter goes over the software implementation of the design and gives a summary of the role of each software package created.
- **Tools** This chapter presents tools created during this project and some examples of their applications.
- **Results and Evaluation** This chapter shows the benchmark results of the CNV detection tools developed. They are evaluated on simulated and real data sets. The results give a reference comparison to the state of the art.
- **Discussion** This chapter globally discusses the project and its results.
- **Materials and Methods** This chapter describes how to repeat the experiments and how to access the data. This chapter also describes the variant comparison and validation methods used in more details.
- **Conclusions** The last chapter concludes this report. It summarizes the results, discussions, and lists the future works.
- **Appendices** The appendices include extra information that is not required for the understanding of this work, but that can be helpful to the researcher. Topics include common file formats for genomics and useful tools and commands when handling genomic related files.

2 Background

Contents

2.1 DNA and the Genome

A genome is defined as the genetic material of an organism, it consists of DNA (deoxyribonucleic acid) and includes coding regions (genes) as well as non-coding regions. The genome is the aggregate of all the hereditary information of the organism. This information, encoded with DNA, is packed into chromosomes. The number of chromosomes in the human genome 46,XX or 46,XY (23 pairs) was determined in 1956 [1]. The chromosomes contain part or all of the genetic material, the genome, of an organism.

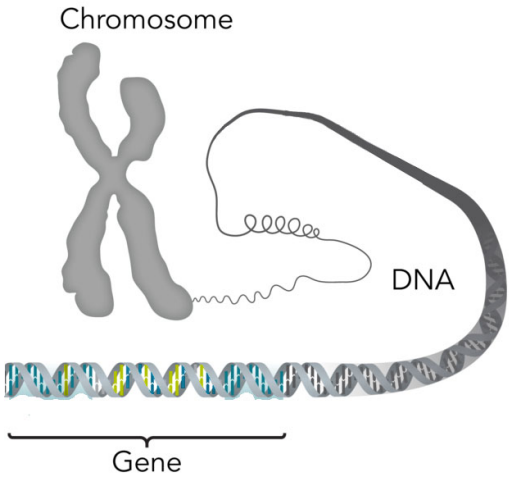


Figure 2.1 Sketch of a chromosome, DNA, and a gene
Image by Thomas Splettstoesser under CC BY-SA 4.0 [2]

2.2 Variation in the Genome

Variation in a genome represents all the differences in DNA content and structure of a genome compared to a reference genome. Variations such as mutations or rearrangements are one of the driving forces behind evolution and differences of phenotypes. Chromosomal recombination during meiosis and subsequent sexual reproduction are a major factor of genetic diversity. Aberrations and disruptions that can occur during these processes are a major cause of genetic conditions in humans. Detections of trisomies (presence of three chromosomes instead of the usual pair) for chromosomes 13, 18, and 21 were tied to clinical phenotypes (Patau, Edwards, and Down syndromes). Theses cases were proof that the copy number of whole chromosomes had repercussions in the expressed phenotypes. In the 1990's, locus specific (subregion inside a chromosome) disorders were discovered, such as Charcot-Marie-Tooth, DiGeorge, Miller-Dieker, Prader-Willi, and Smith Magenis syndromes. The examples showed that it is possible to link variation of copy number in specific loci to specific expressed phenotypes. The advances in DNA sequencing techniques and the creation of the first complete human genome sequence [3] [4] allowed for unbiased detection of variation in individual genomes.

Several forms of variations have been linked to expressed phenotypes but most structural variations such as copy number variations (CNVs) still have unknown, if any, repercussions. Widespread application of whole genome sequencing, and detection of structural variation has shown their broad diversity in type and size. It was also found that they affect more of the genome per nucleotide base changes than any other class of sequence variant [5, 6, 7, 8, 9, 10]. Variations are present in every human genome and affect molecular as well as cellular processes, regulation, transcription, and the 3D structure of proteins [9, 11, 12]. They can play a key role in cellular activity and will have repercussions that range from insignificant to extreme, e.g., play a major role in the viability of the individual. Being able to link variants to these repercussions and expressed phenotypes will help improve the understanding of the genome as well as help improve genetic disease discovery and developments of possible treatments. Therefore, in order to start finding correlations between variants and expressed phenotypes it is of utmost importance to have an efficient and fast way to detect and classify variation. Detection is an important step required to link variation (causes) to phenotypes (consequences). The consequences being e.g., changes in processes and structures in the cell and their resulting changes in physiological and pathophysiological processes.

Variations can occur on the maternal, paternal or both copies of the genome. For example losing a portion of one parental copy at a location will result in a loss of heterozigozity (hemizigozity). The examples shown below will be relative to a single copy (paternal or maternal) but any combination can occur. Relative to both, the maternal and paternal copies of the genome, a loss in a single part, a change in copy number from one to zero, can be interpreted from two to one relative to the complete genome (both parts). Imputations of variations to a single copy can be done in some cases. This can also be facilitated with access to maternal and paternal DNA. However, some variations can be more difficult to phase (assign to the maternal or paternal copy), especially if the mutation is not present in the parents DNA.

Detected variants are traditionally encoded as differences to a reference genome (e.g., the human reference genome). This reference is usually a monoploid reference (single set of chromosomes). Therefore the examples below are relative to a single reference sequence.

2.2.1 Single Nucleotide Variations

Single nucleotide variations (SNVs) or single nucleotide polymorphisms (SNPs) are a substitution of a single nucleotide at a specific location in the genome. They are the smallest form of variation (with single nucleotide insertions or deletions). Figure 2.2 Shows an example of single nucleotide variation.

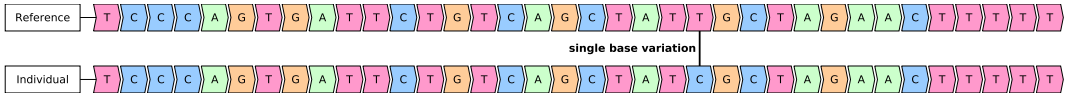


Figure 2.2 Single Nucleotide Variation

Single nucleotide variations have been well studied and linked to phenotype expression. Several detection software and pipelines exist such as the Genome Analysis Toolkit GATK [13], these are mature and come with best practice guidelines. Many analysis pipelines for sequencing data come with SNP discovery stages.

2.2.2 Copy Number Variation

Copy number variation is actually an umbrella term for a subgroup of structural variants. They can be further separated into two categories *unbalanced copy number variants* and *balanced copy number variants*.

Unbalanced Copy Number Variants

Unbalanced variations are variations where the number of bases in the genome will differ from the reference. For example, a deletion, a region where part of the reference genome is not present in the genome of an individual. Unbalanced variations include deletions, insertions, and duplications.

Deletions Deletions are the absence (or loss) of DNA content compared to a reference. Figure 2.3 shows an example of a deletion. The breakpoints, regions where the deletion start and stop can be well defined on the reference as is the case in the figure but may also be more fuzzy because of the process that led to the deletion. Therefore, some deletions have clear cut edges and other actually still contain DNA sequence for a few bases that is different or from to the reference. However in both cases part of the DNA sequence found in the reference is missing.

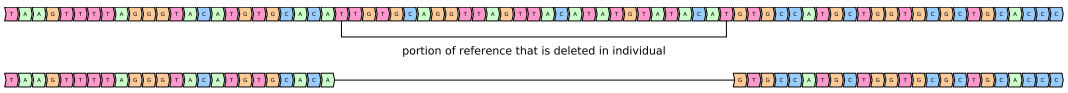


Figure 2.3 Deletion

Insertions Insertions are the dual of deletions and are events where the individual genome contains more DNA than the reference. An example is shown in Figure 2.4 where the sequence "TGTAGGACTGTATAT" appears in the individual genome but not in the reference genome.

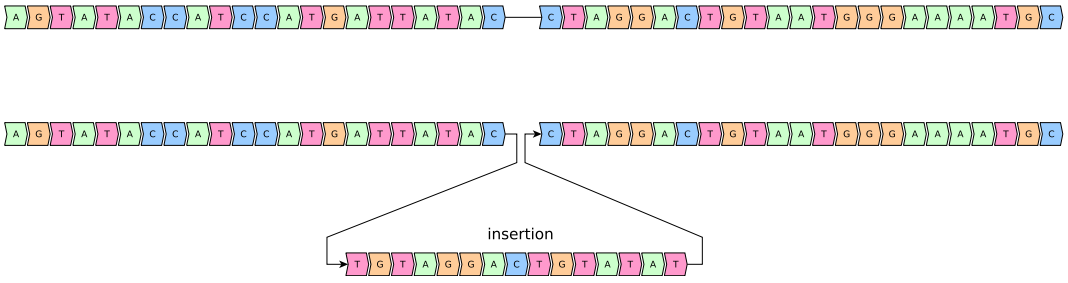


Figure 2.4 Insertion

Duplications Duplications are a replication of a sequence or a gene somewhere in the genome with relation to the reference. These can be repetitions as shown in Figure 2.5 where a sequence is repeated more often in the individual than the reference, here one sequence is duplicated more in one genome than in the other.

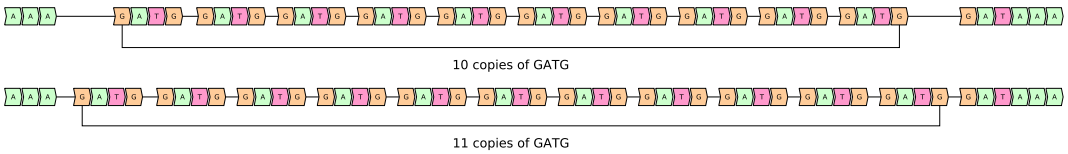


Figure 2.5 Copy number increase due to duplication

Duplications include any replication, or increase in copy number, of a sequence or a whole gene. With genes, their sequence can sometimes differ but still remain functionally similar. Duplications, replication of a sequence, can occur side-by-side to the original sequence, on other loci of the chromosome or even across other chromosomes.

The key difference between insertions and duplications is that duplications insert sequences or genes that were already present in the genome while insertions can be novel sequences. Due to the biological nature of the events and processes that lead to them, the duplications are not always error free, and the insertion of highly similar but different sequences are often still considered duplications. This is of course highly dependent on the nature of the differences, in some cases the change of a single base can knock-off or drastically change the functionality of a gene and these cases require specific attention.

Tandem Duplication A tandem duplication is when part of the genome is duplicated at the location of the original sequence.

Dispered Duplication A dispersed duplication is when part of the genome is duplicated somewhere else in the genome, either on the same chromosome, or on another chromosome.

Balanced Copy Number Variants

Balanced CNVs are variations that do not increase or decrease the number of nucleotides in the genome compared to the reference. Rearrangements are examples of balanced CNVs e.g., inversions and translocations.

Inversions Inversions are events where part of the DNA sequence is inverted as can be seen in Figure 2.6 and 2.7.

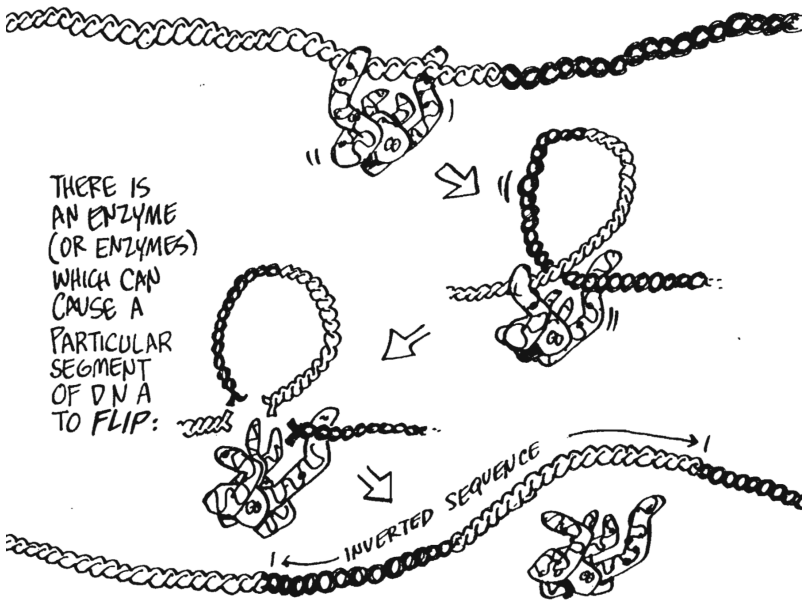


Figure 2.6 Inversion - Illustration by Larry Gonick in The Cartoon Guide to Genetics [14]

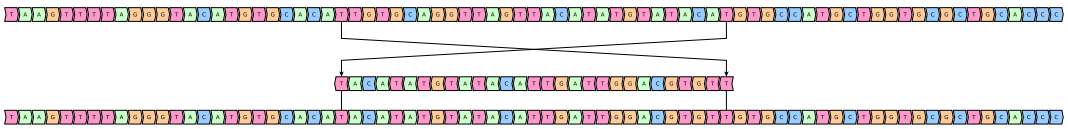


Figure 2.7 Inversion

Inversions play a key role in gene regulation and are part of the cell systematics so they can either be inherited or temporarily present due to the regulation mechanisms in the cell.

Translocations A translocation is when part of the genome sequence is moved to another location compared to the reference. Translocations can be interpreted as a loss (deletion) at some region and a gain (insertion) at another region so sometimes it is not clear if the rearrangement is really balanced. Especially when DNA content is moved to parts of the genome that differ highly with relation to the reference.

Interchromosomal translocations Interchromosomal translocations imply that the part that is moved compared to the reference resides on another chromosome. Interchromosomal translocations can be reciprocal (balanced, Robertsonian translocation (ROB)) meaning that the chromosomes exchange part of their content or non reciprocal, where there is a transfer from one chromosome to another instead of an exchange.

Interchromosomal translocations have been linked to several diseases and conditions. Figure 2.8 shows an overview of some known possible translocations and related conditions.

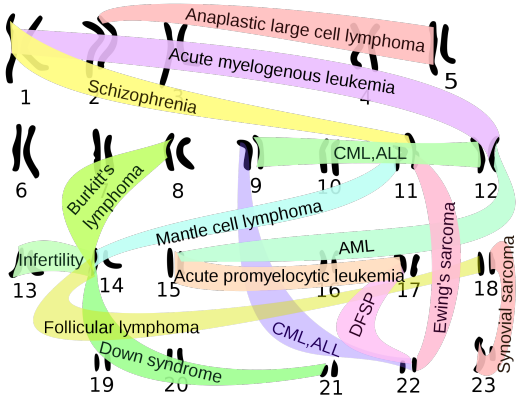


Figure 2.8 *Overview of some chromosomal translocations involved in different cancers, as well as in some other conditions, e.g., schizophrenia, with chromosomes arranged in standard karyogram order. Image by Mikael Häggström (Public Domain) [15]*

Intrachromosomal translocations Intrachromosomal translocations imply that the part that is moved compared to the reference still resides on the same chromosome. This is a combination of a deletion of content at a location and insertion of the same content elsewhere on the same chromosome.

2.2.3 Structural Variation

Structural variation is a supergroup to copy number variation, it further include events such as, mobile element insertions, multi-allelic CNVs of highly variable copy number, and complex rearrangements, which are combinations of multiple events described above. These more complex events are much harder to identify correctly. Particularly complex or very big events may be impossible to solve completely for a given sequencing technology and may require the use of multiple approaches to solve or describe correctly. Some regions in the individual genome may also be too different from the reference to assess correctly relative to the reference used.

2.3 DNA Sequencing

DNA sequencing is the process of determining the nucleotide (nucleic acid, or base) sequence of DNA molecules. It includes any method that is used to determine the sequence of the bases (most often Adenine A, Cytosine C, Guanine G, and Thymine T)[1] in DNA molecules, and of all DNA molecules in the genome in case of whole genome sequencing.

[1] **Expanding the genetic code** : DNA and RNA are naturally composed of four nucleotide bases that form hydrogen bonds in order to pair. Hoshika et al. added an additional four synthetic nucleotides to produce an eight-letter genetic code and generate so-called hachimoji DNA. Coupled with an engineered T7 RNA polymerase, this expanded DNA alphabet could be transcribed into RNA. Thus, new forms of DNA that add information density to genetic biopolymers can be generated that may be useful for future synthetic biological applications [16].

Knowledge of DNA sequences has become indispensable for basic biological research, and in numerous applied fields, such as, medical diagnosis, biotechnology, forensic biology, virology and biological systemics. Comparing healthy and mutated DNA sequences can help diagnose different diseases including various cancers [17], characterize antibody repertoire [18], and can be used to guide patient treatment through personalized precision medicine [19, 20, 21]. Having a quick way to sequence DNA allows for faster and more individualized medical care to be administered, and for more organisms to be identified and cataloged [22, 23, 24, 25].

2.3.1 The Human Genome Project

"The Human Genome Project (HGP) was one of the great feats of exploration in history. Rather than an outward exploration of the planet or the cosmos, the HGP was an inward voyage of discovery led by an international team of researchers looking to sequence and map all of the genes – together known as the genome – of members of our species, Homo sapiens." [26]

The Human Genome Project started on October 1st 1990 and completed in April 2003. It was able to create a sequence with a sampling of over 92% (2.8 billion base pairs) and with over 99.99% accuracy (less than one error in 10'000 DNA bases) [27].

"The Human Genome Project gave us the ability, for the first time, to read nature's complete genetic blueprint for building a human being." [26]

The human reference genome was created using a few of many collected samples (in order to preserve the donors anonymity). The human reference genome is therefore a combination of several individuals and comes not only from a single person. It created the first reference that could be used as a Rosetta stone to compare and encode individual genomes, sequences, and variants.

The timeline of the Human Genome Project can be found at `https://www.genome.gov/human-genome-project/Timeline-of-Events`

The sequencing of this first human genome took 13 years and brought together leaders in the field. The project drove the development of new sequencing technologies. Sequencing is still an ever growing field and has shown a tremendous rise in efficiency. The cost of sequencing has dropped faster than Moore's law over the last years [28].

The next sections will introduce some of the main sequencing technologies.

2.3.2 Sanger Sequencing

Sanger sequencing is based on the selective incorporation of chain-terminating dideoxynucleotides by DNA polymerase during in vitro DNA replication [29, 30]. Developed by Frederick Sanger and colleagues in 1977. This sequencing technique remained the most widely used for approximately 40 years.

Fig. 2.9 shows the main steps of the sequencing process. Although new next generation sequencing techniques that are both faster and cheaper appeared, Sanger sequencing is still used in small scale projects or for validation of results obtained with next generation sequencing because of the high accuracy and possibility to generate longer DNA sequence reads (400 to 900 base pairs) than next generation short-read technologies.

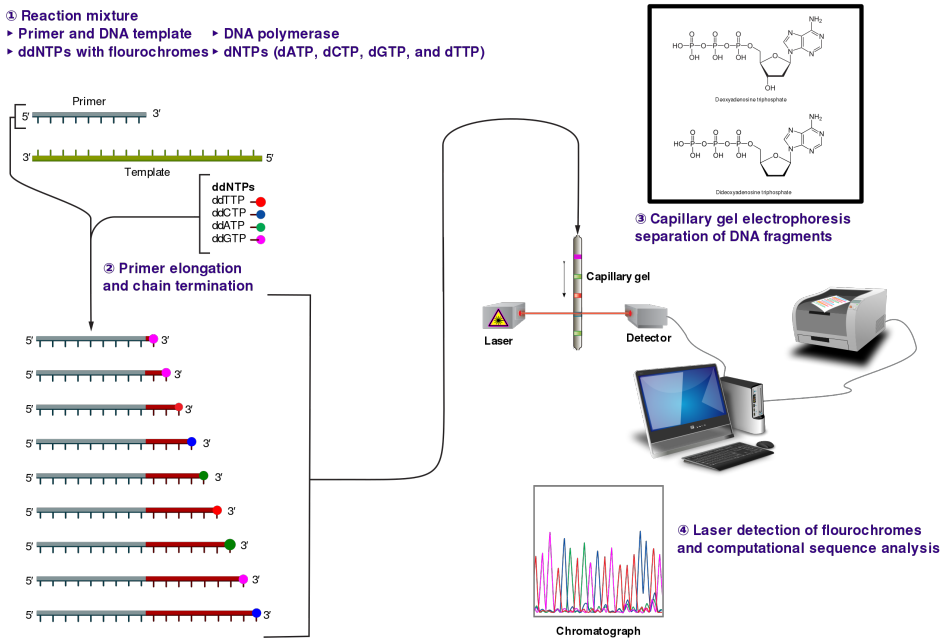


Figure 2.9 *The Sanger (chain-termination) method for DNA sequencing. (1) A primer is annealed to a sequence, (2) Reagents are added to the primer and template, including: DNA polymerase, dNTPs, and a small amount of all four dideoxynucleotides (ddNTPs) labeled with fluorophores. During primer elongation, the random insertion of a ddNTP instead of a dNTP terminates synthesis of the chain because DNA polymerase cannot react with the missing hydroxyl. This produces all possible lengths of chains. (3) The products are separated on a single lane capillary gel, where the resulting bands are read by a imaging system. (4) This produces several hundred thousand nucleotides a day, data which require storage and subsequent computational analysis*
Author: Wikimedia user Estevezj *Source [31]*
Image licensed under Creative Commons (CC BY-SA 3.0) [32]

2.3.3 Next Generation Sequencing

Next Generation Sequencing (NGS) or High-Throughput Sequencing (HTS) are sequencing technologies that parallelized the sequencing process in order to generate thousand up to millions sequences simultaneously, hence the term High-Throughput. The goal with massively parallel sequencing was to reduce the cost and time required by the sequencing process. The short length of the sequenced reads (tens to hundreds of bases) coined the term "Shotgun Sequencing" because of the fast generation of small pieces as with a scattergun shot.

Some of the main short-read, next generation sequencing technologies include :

- Sequencing by synthesis (Illumina)
- Pyrosequencing (454)
- Ion semiconductor (Ion Torrent sequencing)
- Combinatorial probe anchor synthesis (cPAS- BGI/MGI)
- Sequencing by ligation (SOLiD sequencing)

A list of next generation methods and references is available here :

`https://en.wikipedia.org/wiki/DNA_sequencing#Next-generation_methods`

This project focuses on Illumina, sequencing by synthesis paired-end sequencing read fragments as input data.

Paired-end short reads

Paired-end short reads are a couple of sequenced DNA reads originating from a common fragment, both ends of the same fragment are sequenced and the result is a pair of two short reads. This is depicted in figure 2.10.

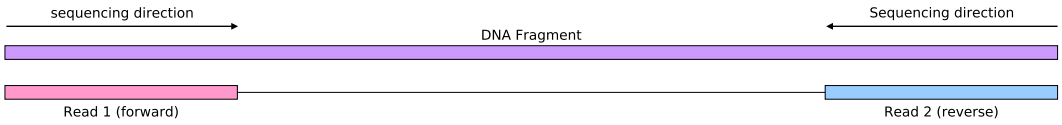


Figure 2.10 Paired-end short reads illustration

The length of the fragments is dependent on the sequencing technology and template preparation but is often in the hundreds to thousands of bases, where the reads themselves are in the hundreds of bases, typically 100-250 bases.

2.3.4 Third Generation Sequencing

Third generation sequencing, also known as long-read sequencing, differs from next generation sequencing (high-throughput sequencing) in that it focuses on reading the base sequence at the single molecule level instead of breaking the long DNA strands into small segments (fragments).

Some of the main long-read, third generation sequencing technologies include :

- Single-molecule real-time sequencing (Pacific Biosciences)
- Nanopore Sequencing (Oxford Nanopore Technologies)

These technologies can generate sequences of tens of thousands of bases albeit at a lower accuracy than short-read next generation sequencing. However, the technology is improving at a rapid pace, reviews can be found in [33], [34], [35], and [36].

2.3.5 Hybrid Solutions

Hybrid solutions came with the idea to reduce the shortcomings of the existing sequencing technologies, e.g., read length or accuracy, by combining different technologies. Sequencing with two or more technologies and combining results is an hybrid sequencing solution.

Polishing long reads

Polishing long reads is the act of reducing the sequencing errors in long reads. This can be done with long reads alone by comparing (aligning) the long reads amongst themselves and taking resulting consensus sequences, if the error is distributed randomly, in most cases, a majority consensus will be the correct sequence. However this can also be done with an hybrid solution, by using short reads to fix the errors on the long reads.

This process is what is called "polishing". The resulting reads or "contigs" (contiguous sequences) are often of the accuracy of short reads (>99.9%) with the length of the long reads. There are however, still cases where it is not possible to correct the errors due to factors such as low complexity of the sequence or high similarity to other existing sequences.

2.4 Read Mapping and Alignment

A sequenced piece of DNA is referred to as a read. A read is a contiguous sequence of bases (nucleotides). For sequencing technologies that create reads from fragments of DNA (long or short) it is necessary to asses where on the genome the fragment originated from. This can be done by multiple techniques, e.g., with markers while sequencing. If the location of origin is not known after sequencing one way to map it back to the reference genome is by comparison, if the sequence is identical or highly similar to a portion of the reference genome, it is highly probable that it came from that region.

Mapping refers to the act of associating a read to a location on a reference genome.

Aligning refers to the act of finding the smallest edit distance between a sequence and another sequence (e.g, a reference genome or subregion thereof).

Mapping and Aligning are often done together because highly similar sequences (or portion of sequences) align well and reciprocally similarity correlates with position of origin (especially when there is a unique location that is highly similar). Alignment being a costly operation, reads are not aligned to the whole genome sequence but rather first mapped to one or multiple probable locations and then aligned to a subsequence of the reference at that location. Since the two operations are often done in tandem we will use *mapping* and *aligning* interchangeably in the rest of this document. A mapped

read implies it is also aligned (not necessarily the best alignment). An aligned read to a reference implies it is also mapped to a position on the reference. Figure 2.11 shows an illustration of reads mapped to positions relative to a reference sequence.

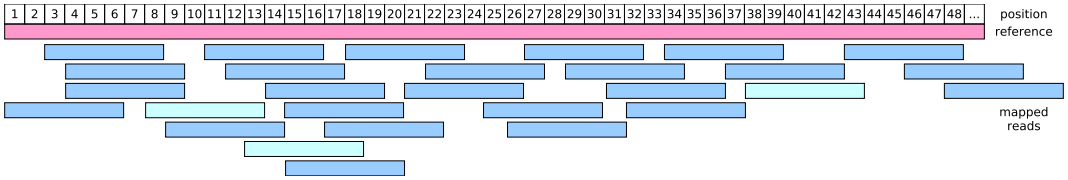


Figure 2.11 *Reads mapped onto a reference, light colored reads imply that they have more than a single possible mapping location on the reference*

Numerous downstream analyses require the reads to be aligned to a reference. Therefore mapping and aligning is one of the most common tasks after sequencing. The mapping and alignment also provides information about the density of reads related to specific regions of a reference as well as their similarity (thus also their differences). Irregularities in alignment will hint at differences between the sequenced sample and the reference used.

2.4.1 References

There are multiple references for the human genome. These references are assemblies created to be a common human genome used as reference point in studies. By using a reference and encoding individual genomes as differences to a reference, individual references can then be compared by their differences to the reference. This removes the need to compare every individual genome against each other in larger or subsequent studies.

The references include contigs (contiguous sequences) for all chromosomes as well as for the mitochondrial chromosome, and may contain unplaced contigs. The main references used as of 2020 are GRCh37/hg19 and GRCh38/hg38 and sequencing data is often aligned to one of them or both depending on the study.

Timeline of main references

The information provided below for the main references comes from the UCSC Genome Browser Gateway and is available at :

`https://genome.ucsc.edu/cgi-bin/hgGateway`

- **NCBI34/hg16** - July 2003

  The July 2003 human reference sequence (NCBI Build 34) was produced by the International Human Genome Sequencing Consortium.

This sequence covers about 99 percent of the gene-containing regions in the genome, and has been sequenced to an accuracy of 99.99 percent. Of note in this release is the addition of the pseudoautosomal regions of the Y chromosome. This sequence was taken from the corresponding regions in the X chromosome and is an exact duplication of that sequence.

There are 2,843,433,602 finished sequenced bases in the ordered and oriented portion of the assembly, which is an increase of 0.4 percent, or approximately 11 Mb, over the Build 33 assembly. The reference sequence is considered to be "finished", a technical term indicating that the sequence is highly accurate (with fewer than one error per 10,000 bases) and highly contiguous (with the only remaining gaps corresponding to regions whose sequence cannot be reliably resolved with current technology). Future work on the reference sequence will focus on improving accuracy and reducing gaps in the sequence.

- **NCBI35/hg17** - May 2004

  The May 2004 human reference sequence (NCBI Build 35) was produced by the International Human Genome Sequencing Consortium.

- **NCBI36/hg18** - Mar. 2006

  The March 2006 human reference sequence (NCBI Build 36.1) was produced by the International Human Genome Sequencing Consortium.

  This assembly contains alternate haplotype regions. For example an alternate chromosome 22 assembly that contains the CYP2D6 gene (NT_113959.1) that was deleted in the reference assembly. The assembly also contains haplotypes for chromosomes 5 and 6.

- **GRCh37/hg19** - Feb. 2009

  The February 2009 human reference sequence (GRCh37) was produced by the Genome Reference Consortium.

  In addition to the "regular" chromosomes, the hg19 contains 9 haplotype chromosomes, 39 unplaced contigs, and 20 unlocalized contigs.

- **GRCh38/hg38** - Dec. 2013

  The GRCh38 assembly is the first major revision of the human genome released in more than four years. As with the previous GRCh37 assembly, the Genome Reference Consortium (GRC) is now the primary source for human genome assembly data submitted to GenBank.

  Several human chromosomal regions exhibit sufficient variability to prevent adequate representation by a single sequence. To address this, the GRCh38 assembly provides alternate sequence for selected variant regions through the inclusion of alternate loci scaffolds (or alt loci). Alt loci are separate accessioned sequences that are aligned to reference chromosomes. The GRCh38 initial assembly con-

tained 261 alt loci, many of which are associated with the LRC/KIR area of chr19 and the MHC region on chr6. Subsequent GRC patch releases have added additional alt loci and fix patches. See the sequences page for the latest list of the reference chromosomes, alternate, and patch sequences in GRCh38.

Debuting in this release, the large megabase-sized gaps that represented centromeric regions in previous assemblies have been replaced by sequences from centromere models created by Karen Miga et al. [37], using centromere databases developed during her work in the Willard lab at Duke University (now at the University of Chicago) and analysis software developed while working in the Kent lab at UCSC. The models, which provide the approximate repeat number and order for each centromere, will be useful for read mapping and variation studies.

The mitochondrial reference sequence included in the GRCh38 assembly has been updated.

Several erroneous bases and misassembled regions in GRCh37 have been corrected in the GRCh38 assembly, and more than 100 gaps have been filled or reduced. Much of the data used to improve the reference sequence was obtained from other genome sequencing and analysis projects, such as the 1000 Genomes Project.

The GRCh38 assembly offers an "analysis set" that was created to accommodate next generation sequencing read alignment pipelines. To avoid false mapping of reads, duplicate copies of centromeric arrays and WGS on several chromosomes have been hard-masked with Ns. The two PAR regions on chromosome Y have also been hard-masked, and the Epstein-Barr virus sequence has been added as a decoy to attract contamination in samples. Two versions of the analysis set are available on our downloads page: one without the alternate chromosomes from this assembly, and one that includes them.

For this projects GRCh37/hg19 will be used because this was the reference used by the Genome Center to align their sequencing data. Also because most of the benchmarks provided by Genome in a bottle (see section 2.5 below) are provided for GRCh37. This project will however be developed in a way that is reference agnostic, the same techniques will work for other or newer references.

2.4.2 Mappers / Aligners

There exist numerous software that do the mapping and alignment of sequencing reads to a reference. Some commonly used aligners include the Burrows Wheeler Aligner (BWA), the Isaac Aligner, Novoalign, BBMap, Bowtie, and others.

Data for this project will already be mapped and aligned. By BWA to hg19 in case of data provided by the Genome Center and sometimes by other aligners for data provided by other sources.

An extensive list can be found at `https://en.wikipedia.org/wiki/List_of_sequence_alignment_software` and this list provides references for all the different mappers and aligners as well as references to benchmarks.

2.4.3 The Integrative Genomics Viewer - IGV

The Integrative Genomics Viewer (IGV) is a high-performance visualization tool for interactive exploration of large, integrated genomic datasets. It supports a wide variety of data types, including array-based and next-generation sequence data, and genomic annotations [38, 39, 40].

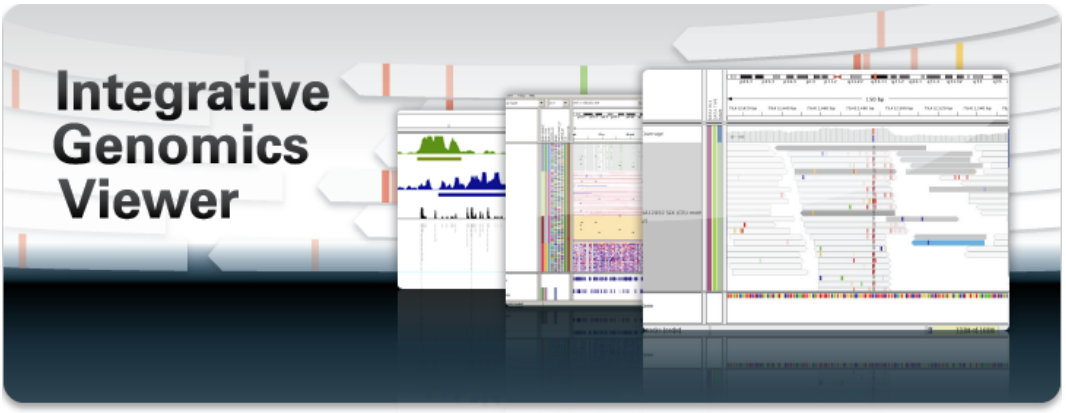


Figure 2.12 *IGV Banner © 2013-2018 Broad Institute and the Regents of the University of California Source [41]*

This application allows, amongst others things, to see aligned reads on a reference genome as can be seen in figures 2.13 and 2.14. Examples of variation in the genome in chapter 3 will be illustrated with screen-shots of IGV. IGV will be used for validation and visualization of genomic data throughout this project and is in itself an invaluable tool for genomic data analysis.

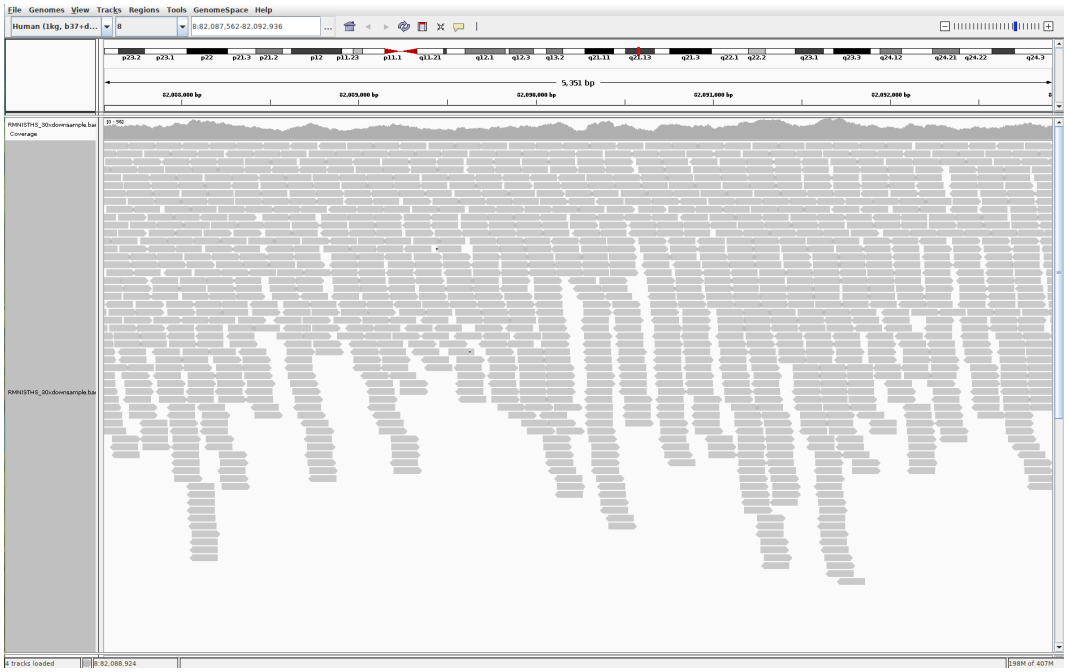


Figure 2.13 Reads mapped to a reference in IGV

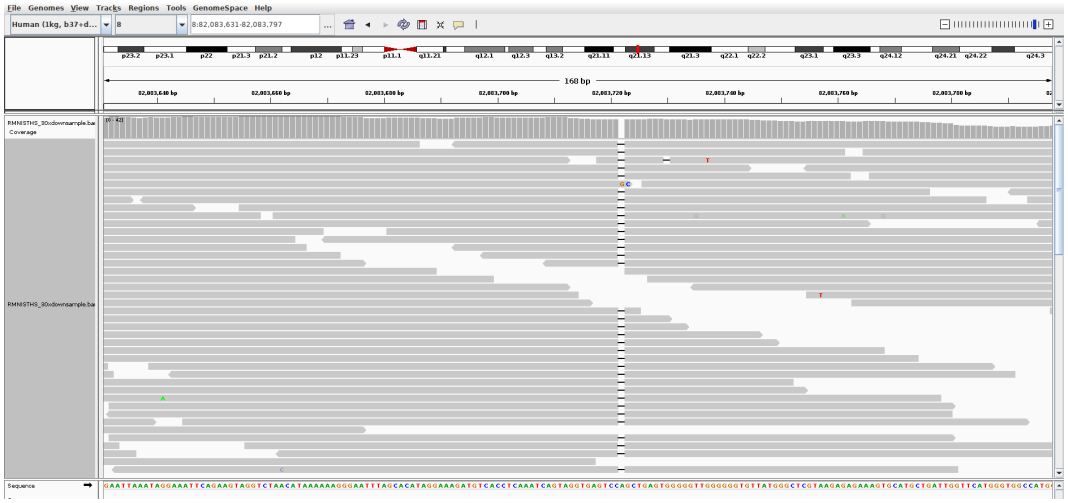


Figure 2.14 Reads mapped to a reference in IGV with visible result of alignment (1 base gap)

2.5 Genome In A Bottle

With the existence and development of multiple sequencing technologies came a need for a common ground for comparison. Here is where the Genome In A Bottle (GIAB) project comes in. The project provides reference samples that can be used to benchmark and qualify sequencing technologies and platforms. This is of importance because without common samples, direct comparisons of sequencing methods and data would be difficult if not impossible. The main services and data provided by GIAB are summarized below.

Figure 2.15 GIAB logo Credit NIST [42]

The Genome In A Bottle (GIAB) project is a consortium comprised of public, private, and academic entities. It is hosted by the US National Institute of Standards and Technology (NIST) to develop the technical infrastructure to enable the translation of whole human genome sequencing to clinical practice [42].

2.5.1 Provided by The Genome In A Bottle consortium

The GIAB consortium provides the following

Reference samples GIAB has currently characterized a pilot genome (NA12878 / HG001) from the HapMap project [43], and two son/father/mother trios of Ashkenazi Jewish and Chinese ancestry from the Personal Genome Project [44] (selected because, unlike the pilot genome, they are consented for commercial redistribution and reidentification) [42].

Benchmark variant calls and regions GIAB also provides "high-confidence" variant calls and regions. These were generated by several sequencing technologies and can be used to benchmark and validate new variant calling pipelines. Small variants references files (in VCF and/or BED formats) are available for the two reference genomes GRCh37 and GHCh38.

Draft benchmarks for difficult variants and regions GIAB also provides variant calls and regions for more difficult calls such as, bigger structural variants or small variants in difficult regions.

Benchmarking best practices GIAB worked with the Global Alliance for Genomics and Health Benchmarking Team to create best practices and benchmarking tools [45]. The tools for variant comparison and evaluation are a collection of scripts avaible publicly at `https://github.com/ga4gh/benchmarking-tools/`.

Sequencing Data Datasets of diverse sequencing technologies (short-reads, long-reads, linked-reads, and others) are publicly made available and without publication embargo. This allows researchers to analyze sequencing data coming from different technologies and run new analysis pipelines on known and well studied samples. This will be the main source of data for the benchmarks of this project. The sequencing data is available at `ftp://ftp-trace.ncbi.nlm.nih.gov/ReferenceSamples/giab/data/`

3 | Analysis

Contents

3.1 Introduction

This section presents the analysis of variation detection techniques based on sequencing data, more specifically short-read paired-end sequencing data. It also introduces a list of variation databases and existing software related to CNV or SV detection and analysis.

3.2 Structural Variation Detection and Analysis

There are several of methods to detect structural variation in a genome, such as fluorescence in situ hybridization (FISH) or array comparative genomic hybridization (aCGH). This work will however focus on sequencing based methods more specifically with next-generation short-read sequencing based (Illumina) as input data. This chapter will go over the main techniques used by existing software to detect variation in the genome.

3.2.1 Single Nucleotide Variations and Small Indels

Single nucleotide variations can be considered as the smallest structural variations, i.e., the loss of one base replaced by another. There are extensive studies of SNVs in the human genome and there exist multiple of detection pipelines and best practices for SNV calling. It is not the goal of this project to detect SNVs. They are introduced here for completeness.

SNVs are detected by the analysis of aligned reads to the reference genome, in a nutshell, if enough mapped reads support a SNV (have the mutation) at a given position in the genome, it is marked as such.

Small "indels" are small insertions or deletions less than 50 bases long. They are often referred to as "small indels" or simply "indels" in contrast to CNVs or SVs which are defined as sequence variations of more than 50 bases long. This is arbitrary and we will consider "indels" as CNVs in the remainder of this text. If the distinction is to be made between small indels and CNVs it can be done during interpretations of the CNV calls. Events of the insertion or deletion type smaller than the 50 base threshold can be classified as indels instead of CNVs.

3.2.2 Copy Number Variations

The different classes of copy number variations were introduced in the previous chapter and this chapter will focus on the classes of algorithms used to detect them using short-read sequencing data.

3.3 Introduction to Variation Detection

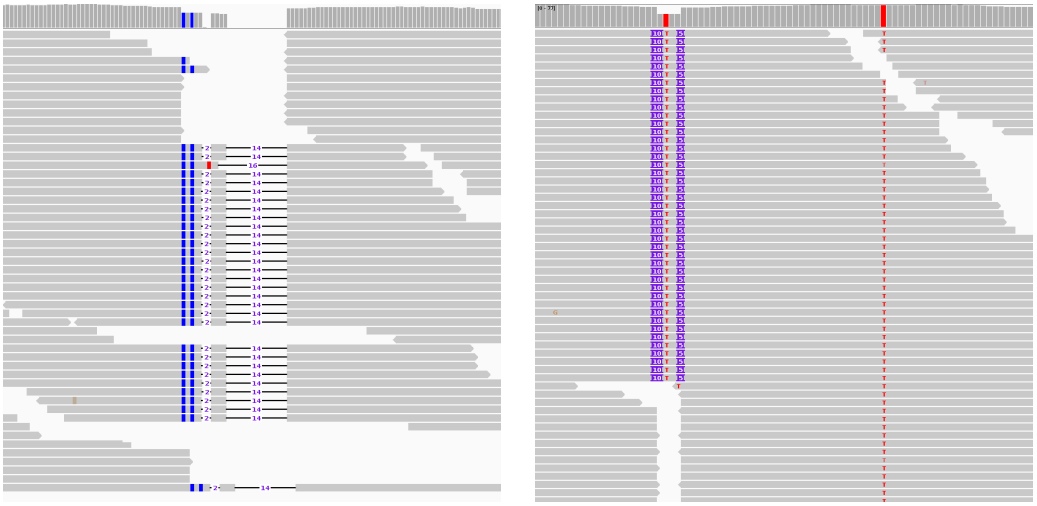
Events, variations, in an individual genome relative to a reference genome will generate differences in the sequenced reads as well as a different global distribution of sequenced DNA read on the reference sequence. By remapping the sequencing reads to the reference and by analyzing their contents it is possible to discover many small variations

(indels), copy number variants, as well as hints to larger or more complex structural variations. In this section we will discuss how the mapped reads can be used to infer structural variation.

The examples will be accompanied by real sequencing data shown in the Integrated Genome Viewer IGV [41].

3.3.1 Small variations detection

Small variations, SNVs and indels of less than 50bp can often be directly inferred from the aligned sequencing reads themselves. Figure 3.1 shows examples of small variations directly visible on the aligned reads. These can be detected by analyzing the alignments of reads directly, however in some cases it can be more challenging, especially due to errors in sequencing.



(a) Small deletions and SNVs *(b) Small insertions and SNVs*

Figure 3.1 Small reads (150bp) spanning over variation events

The strategy applied for detection and resolution of SNVs and small indels in the popular HaplotypeCaller tool [46] of GATK [13] is to traverse the aligned reads and when enough reads differ from the reference they are extracted and a local reassembly is applied to generate candidate sequences called haplotypes (haploid genotypes). The candidate sequences (haplotypes) are then scored based on their likelihood given the sequencing reads and the most likely sequences are given for the region. The differences in these sequences compared to the reference hold the variations.

3.3.2 Copy Number Variation detection

This section will discuss how properties of aligned short-read sequencing data are used to detect variation or variation breakpoints.

Several reviews and studies on detection software and algorithms have come up with classification schemes for the algorithms and techniques used. The naming scheme for the algorithms can differ slightly from publication to publication. For this text the naming used will be the same as in [47].

Individual Methods

This section shows methods that rely on a specific property of the sequencing data to predict or extract regions that exhibit signs of structural variation.

- **Read Depth** : Read Depth, also known as Read Count or Coverage corresponds to the number of reads that map to a position in the reference genome. For example if no reads map to the reference genome at some position it usually means this portion of the genome is not present in the individual genome (i.e., a deletion). The average read depth over the genome is relative to the sequencing depth. Distribution of reads will be dependent on the sequencing methods and specific regions of the genome but the average depth is still relative to the sequencing depth. An example of aligned reads with read depth graph (at the top) can be seen in Fig. 3.2.

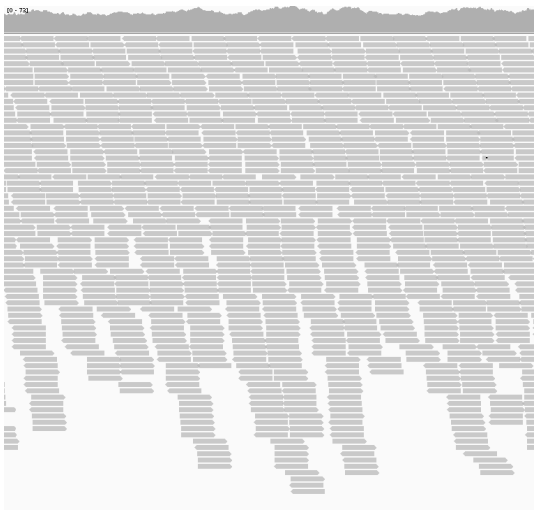
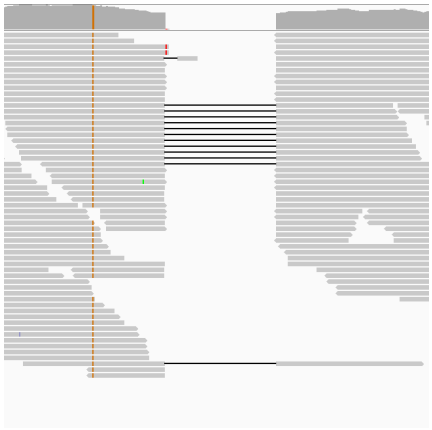
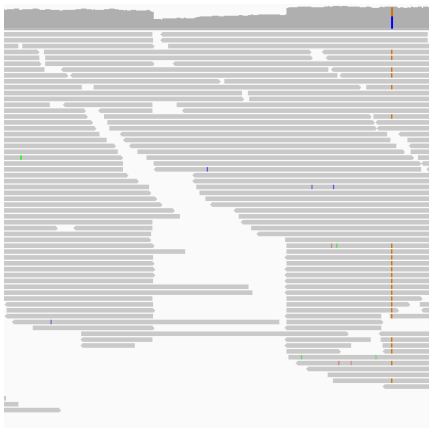


Figure 3.2 Sequencing reads aligned to a reference genome with read depth graph

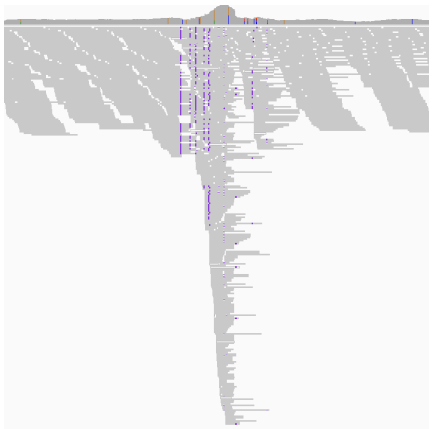
This information can be used to find structural variants. Figure 3.3 shows several examples of structural variations and their impact on read depth. These cases were chosen for their obvious impact on read depth and other forms of structural variation may have a more subtle effect on read depth. Read depth is nevertheless strongly correlated with copy number, if a region is absent in the individual genome it will not be covered by sequencing data on the reference genome if a region is duplicated compared to the reference it will, in general, lead to higher read depth when the sequencing reads are realigned on the reference. Figures 3.3a and 3.3b show a homozygous and heterozygous deletion respectively. Figures 3.3c and 3.3d show examples of duplications and the resulting increase in coverage.



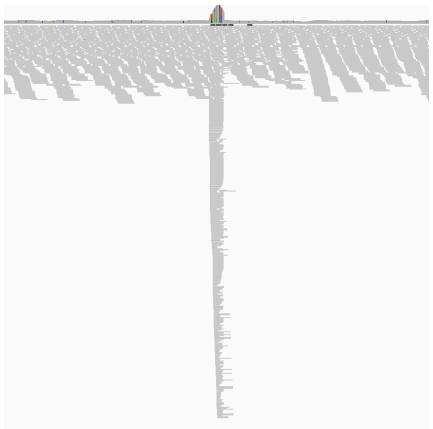
(a) Deletion



(b) Deletion on a single allele



(c) Duplication / Triplication



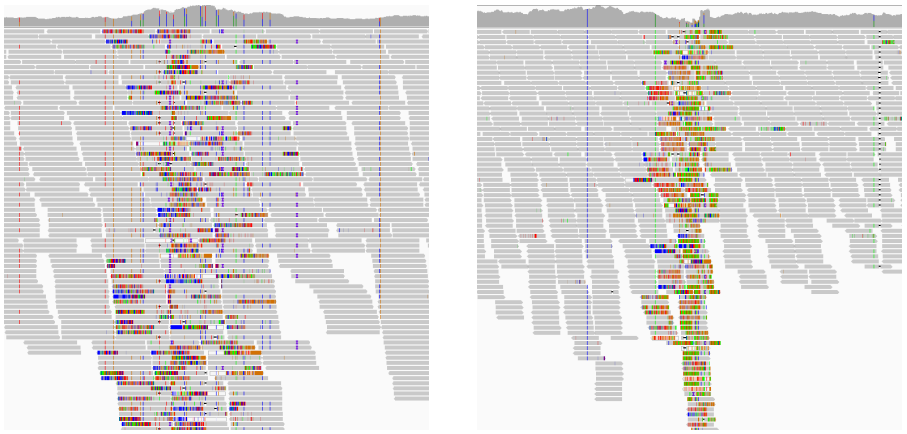
(d) Multiple duplications

Figure 3.3 Examples of structural variation with read depth shown

The read depth is not always representative, for example there are regions of the genome that have a low "mapability" which will affect the read depth. This can be because of low complexity or self-similarity which will result in reads not being mapped correctly or possibly mapped to multiple positions. There is also the problem that several regions in the genome are similar and reads may be mapped incorrectly or at least inconclusively.

- **Mapping discordance** : Mapping discordance is when reads are mapped to the reference but exhibit mismatches in their alignment. In case of mutations such as SNVs the particular base that is different will be discordant to the reference. This is also the case for small indels. With insertions of novel DNA content or more complex events, mapping discordance is often high. This is also the case around breakpoints of any variation, because this is where the individual genome begins to differ compared to the reference and reads that span these breakpoints will exhibit many difference in their half that overlaps the variation. Figures 3.4a and 3.4b show examples were the mapping discordance is high. Mismatched bases are shown colored. Mapping discordance being the differences between the

sequencing data and the reference region where they are supposed to originate from is directly related to variations. This can be for small variations as shown in section 3.3.1 or for bigger or more complex events such as shown here.



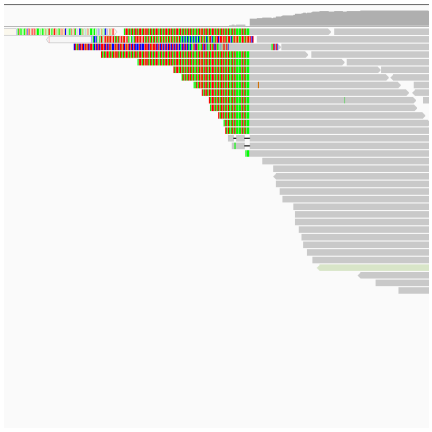
(a) Duplication/Insertion event with high mapping discordance

(b) Complex event with high mapping discordance

Figure 3.4 Examples of structural variation with high mapping discordance

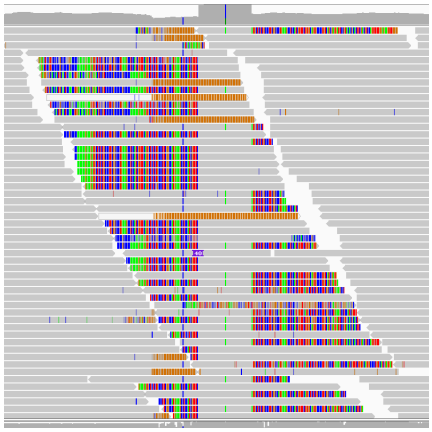
- **Split Reads (Clipped Reads)** : Split reads are reads where part of the read will map to one location of the genome and the other half to another location (or not map at all). The mapping will depend of the software used to generate the alignment. Some software will mark multiple alignments, some will not. Note that reads that map completely to multiple loci of the genome are not split reads.Aligners will also often map the reads to a location but clip the other half of the read. Therefore a read where only half of it is mapped can also be considered a split read. Reads that overlap the start of end of a variation will usually end up as split reads when aligned. When one end is mapped, the bases of the other end will be marked as clipped even when they are aligned elsewhere. Therefore when there is a split read there will be clipped bases and clipped bases mean only part of the read is mapped at the current location. Therefore it is often more convenient to check if a read is clipped when looking for variation breakpoints then to check for multiple alignments.

  Figure 3.5 shows examples of split (clipped) reads around variation events. The split portion of the read (portion that maps elsewhere or does not map at all) is shown with colored bases since they don't match locally.

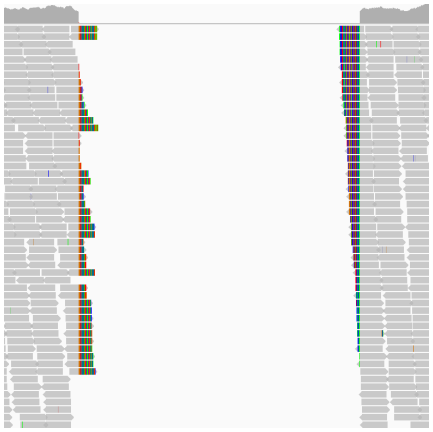


(a) Edge of a deletion event with locally mapped end of the read in gray and split part colored (missmatched bases)

(b) Same location but with only the locally mapped end shown (reads have been rearranged with relation to their vertical ordering)



(c) Insertion with split (clipped) part of reads colored

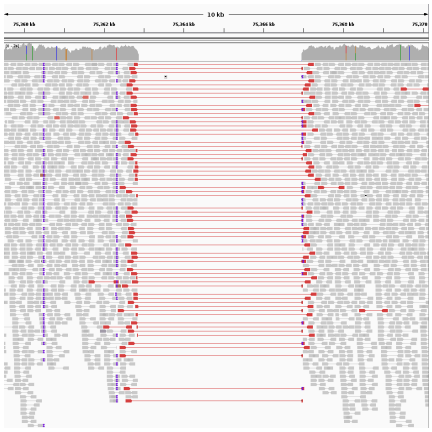


(d) Deletion with split (clipped) part of reads colored

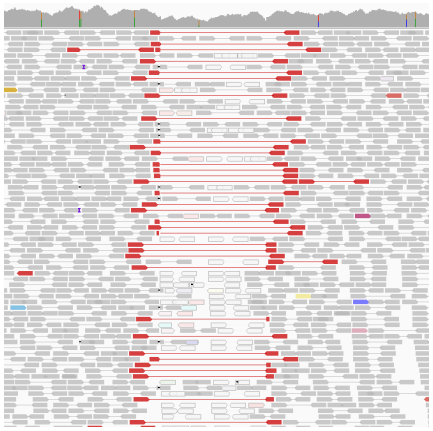
Figure 3.5 Examples of split reads around variation sites

- **Distance** : Paired-end sequencing technology reads a single DNA fragment from both ends resulting in two paired sequences (two reads). The size of the fragment and sequenced ends is dependent on the sequencing technology as well as the library preparation. With NGS the ends are often of a few hundred base (150-250bp) and the fragment sizes are from a few hundreds to a thousand bases. Because this depends on multiple parameters the fragment size distribution is computed from the aligned sequenced reads. Fragments are expected to have a length close to the average length of the distribution. If their length inferred from the alignment is much higher than the expected length, either the reads are misaligned or there is a form of variation in the genome. The same applies for the cases where the inferred length is too small e.g., smaller than the sum of each end or even negative. In some cases where one end is mapped to a position the other end can be mapped to an entirely different chromosome or not be mapped at all. In these cases the distance cannot be computed but this can also be sign of structural variation, e.g., an interchromosomal translocation.

Figure 3.6a and 3.6b show two deletions, one homozygous (present on both alleles), the other heterozygous (present on only one allele). The sequencing fragments that span the deletion have their ends mapped to each side of the deletion. The inferred fragment size from the alignment is much higher (marked red) than the average fragment size and therefore indicates that both ends, both reads, are actually closer to each other on the individual genome than on their mapped position on the reference genome.



(a) >4kbp Deletion (inferred fragment size too big)

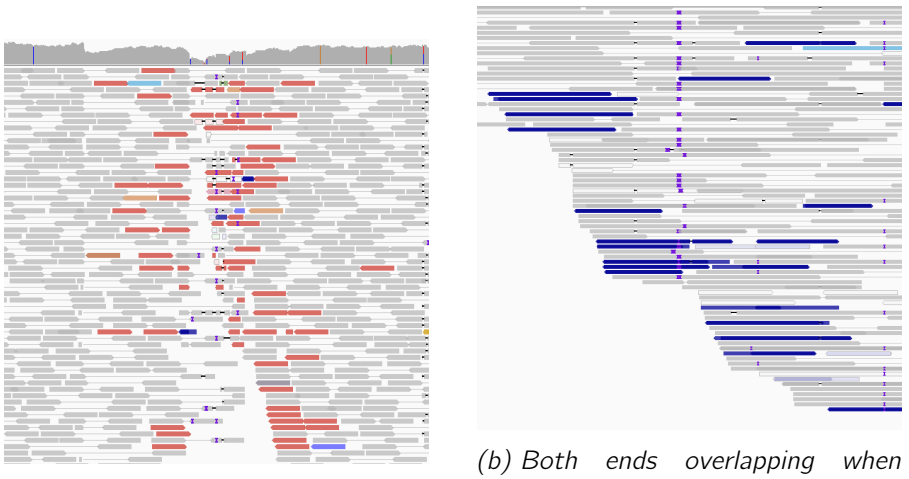


(b) Deletion on a single allele, hard to identify with read depth

Figure 3.6 Examples of alignments with abnormal (too big) inferred fragment size

Figure 3.7a shows reads, marked orange, that have their mate (other end of the fragment) mapped to another chromosome. Therefore, their inferred fragment distance is incorrect because in reality the fragment originates from a single chromosome. Fragments that have one end mapped to a chromosome and the other mapped to another can either indicate the ends are mapped incorrectly, or that there is a form of variation such as an interchromosomal translocation, chromosomal fusion, insertion of novel DNA sequence that maps better to another chromosome, mobile element insertion, etc.

Figure 3.7b shows fragments were both ends overlap when mapped to the reference (marked in dark blue). This is often sign of an insertion or duplication. For example, if a fragment spans a region that is duplicated in the individual genome but not the reference, both ends may cover a different copy of the same sequence but when remapped to the reference their mapped position will collapse on the single copy available on the reference.



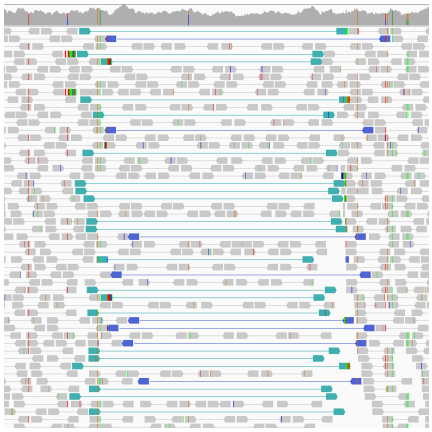
(a) Mates of reads marked orange are mapping to other chromosome

(b) Both ends overlapping when mapped to reference in blue (inferred fragment size too small)

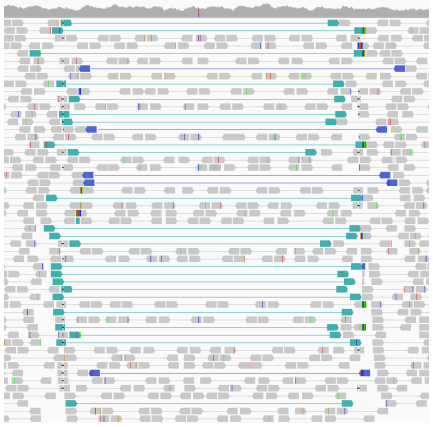
Figure 3.7 Examples of alignments with abnormal inferred fragment size

- **Orientation** : With paired-end read sequencing technology (Illumina) the DNA fragments are read from both ends resulting in a paired-read fragment. Because the sequencing (reading) process goes from the outside of the fragment towards the center, one read is sequenced in the reverse direction compared to the other (one end is read in the 5'-end to the 3'-end direction and the other end is read in the 3'-end to 5'-end direction). The opposing direction of sequencing at each end means that when they are mapped back to the reference, one end should be in the forward direction and the other in the reverse direction. Because the direction of the two ends is physically imposed, when a mapped pair has an unexpected orientation it can either mean that the ends are mapped incorrectly or that one end comes from a sequence that is inverted, or part of a more complex variation in the individual compared to the reference.

  Figures 3.8a and 3.8b show examples of inversions in the individual genome marked by the presence of read pairs with improper orientation. The inferred fragment size (too big) is also a result of the inversion. For all the other fragments that originate from inside or outside the inverted sequence, they map back normally to the reference genome. Only the fragments that span the breakpoints around the inversion sites are mapped with improper orientations and fragment sizes (distance between reads) on the reference.



(a) Inversion example with improper read orientations

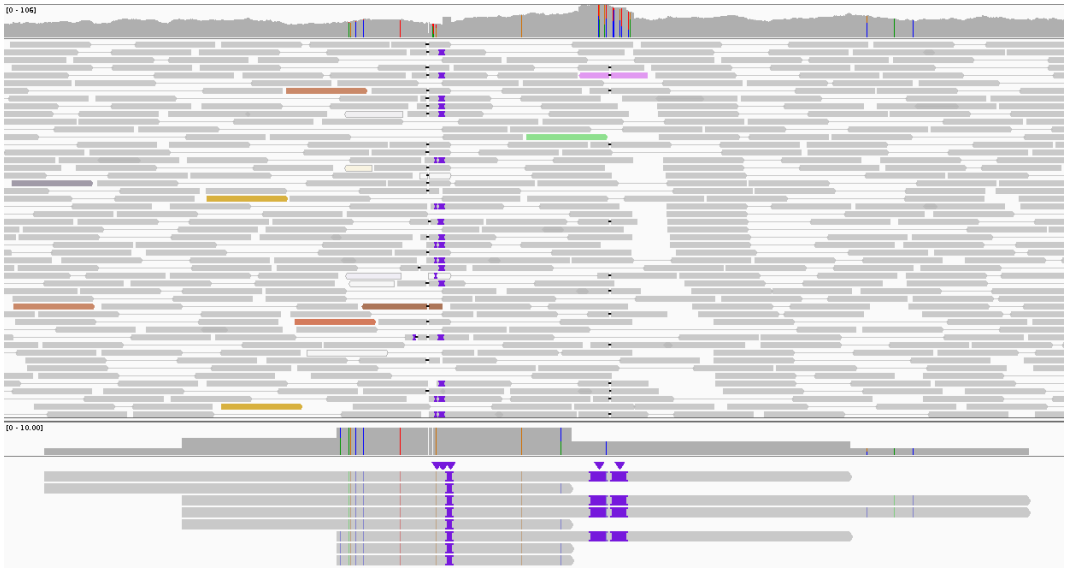


(b) Inversion example with improper read orientations

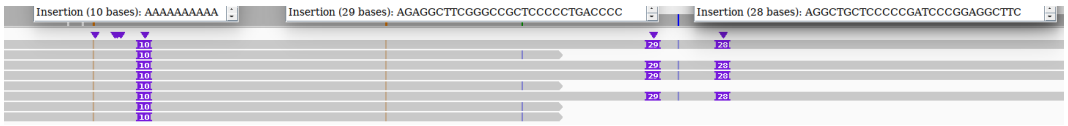
Figure 3.8 *Examples of inversions with improper read orientation, forward-forward in teal and reverse-reverse in blue, normal forward-reverse in gray*

This property is very helpful for the detection of inversions and as can be seen in the figures. The impact on coverage (or other properties) is not significant. There are also split reads (clipped reads) around the breakpoints which could help detection too.

- **De-Novo Assembly** : Reassembling parts of the genome from aligned or unaligned sequencing reads, then mapping the assemblies back to the reference also makes it possible to discover structural variation. One example where this technique helps over the other alignment based techniques shown above is to detect and solve novel insertions. The other techniques above may help identify breakpoints in the genome but they are not enough to solve all cases. For example, a novel insertion of DNA material that is not present in the reference will generate unmapped reads in the aligned sequenced reads as well as anomalies in the read depth, distance or alignment of reads. These anomalies may indicate a breakpoint, however they will not allow to tell the size of the insertion let alone its contents. An assembly, if big enough that it spans the event and around can be remapped to the reference. This realigned sequence will exhibit the possible details of the event. Fig. 3.9 shows an example of assembled sequences aligned back to the reference. The assembly allowed to detect the presence of three insertions as well as to find the sequence of the insertions as shown in figure 3.9b.



(a) Sequencing reads (top track) and assembled sequences (bottom track) aligned to the reference



(b) Zoom-in : Three insertions with exact location and contents

Figure 3.9 De-Novo local assembly example

- **Known Variants** : Multiple databases of variants have been developed (e.g., for population scale studies) categorizing and storing recurring and rare variants. These can be used to determine regions of the genome that are more prone to variation, giving insights as to where to look for variation. A variation could also be common in a population and depending on the individual, this information can be used to help detection. This is mostly a heuristic that has to be complemented by the above techniques in order to asses the variation for the individual. Having variant databases and linking them to known phenotypes helps to do targeted detection. The more and more variants that will have been detected and studied in individuals and populations, the more the databases will grow and help further studies of variation.

- **Other techniques** : Some other characteristics of the reads correlate with variation such as the number of errors in the alignments (edit-distance). It may also possible to search for the presence of known motifs that arise in translocation sites or guide the discovery to sites that are known for instability.

Ensemble Algorithms

Ensemble algorithms are methods that combine two or more of the individual methods above, most current methods are based on ensemble algorithms. Ensemble algorithms apply individual detection methods and choose a way to combine the individual results. For example, an ensemble algorithm could take the union of all the results of the individual methods. Another example would be to filter the results of one method based on the results of another.

Any combination of individual detection methods can be considered an ensemble algorithm. The strategies discussed in the next chapter are based on ensemble algorithms.

Meta Algorithms

Meta algorithms take several existing algorithms or variation calling software and combine their results based on some meta-level heuristic. An example of a meta algorithm is Lumpy [48] which is a probabilistic framework that not only can take individual methods for detection, but also results from other software, as well as known variants in order to call the final variants. Another example is MetaSV which uses multiple callers and uses a trained predictor to call the final variants [49]. As a final example, CN-Learn is a framework that extracts features from a truth set and uses theses features to train a random forest classifier to mark the results of other calling algorithms as true of falls calls [50].

The downside of these approaches is that they cannot predict new variants compared to the union of the calls of the individual software or algorithms.

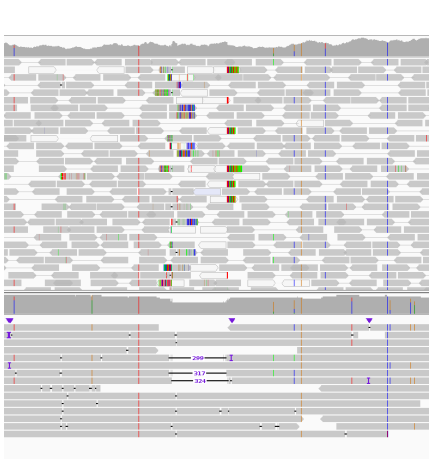
3.3.3 Detection using long reads

Detection of structural variants using long reads of tens of kilo-bases in length can help detect and solve structural variation because the reads may span the entire event. The long reads benefit from their size but come as single reads instead of pairs, therefore extracting inversion information may be more difficult, unless the inversion is small compared to the read itself.

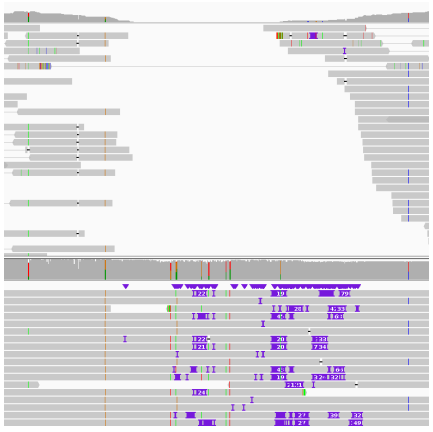
Long reads can be very helpful for the detection and resolution of structural variation that is small compared to their respective length. They can for example span and therefore solve smaller duplication sites completely. They are also better suited for assembly because of their length. The bigger the length of a read the more it is possible to solve with. However, this only pushes the boundaries further. The same problems occur but with bigger lengths (e.g., repetitive regions much bigger than the long reads). In any case, the longer the reads the more it is possible to solve with.

Figure 3.10 shows examples of variations with short reads in the top section and long reads in the bottom section. Figure 3.10a shows a deletion on a single allele that is not directly obvious based on the short reads but is directly visible on the long reads. Figure 3.10b shows insertions that are hard or impossible to solve with short read sequencing. In this case when looking at the short reads, no single read is mapped in some parts of that region. The reads from that region are probably either mapped wrongly elsewhere or left as unmapped reads. Figure 3.10c shows an insertion with

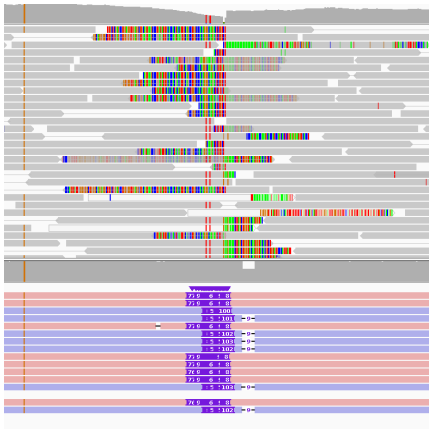
visible impact on short reads. The long reads directly show the insertion, the position of the insertion is dependent on the aligner, here we can see that the aligner used will put the insertions more to the left on forward aligned strands (red) than on reverse aligned strands (blue). The exact insertion position is dependent on the alignment and may not be clear. However the read will contain the exact global sequence and can be analyzed independently of the alignment. Figure 3.10d shows an example of a complex region where the long reads may not be enough to solve the variation. The same problems occur than with short reads but at a larger scale.



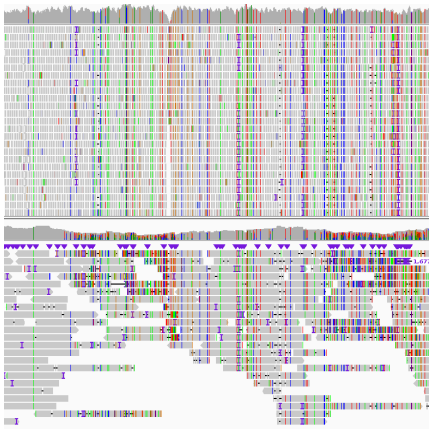
(a) Deletion on a single allele directly visible on long read alignment



(b) Insertions visible on long read alignment, difficult if not impossible to solve with short reads



(c) Insertion visible on long reads, with visible impact on short reads too



(d) Complex region that is hard to solve even with long reads

Figure 3.10 Examples of variations with short and long reads

Long reads allow for better detection and analysis of insertion variant cases but the added cost of an extra sequencing technology and extra required power to process (e.g., polish or align to reference) the reads as well as their higher sequencing error rate may render this approach unpractical.

For this project long reads are not considered as input for detection since the goal is to do CNV assessment from next-generation sequencing data, more specifically Illumina short reads. They were however used to validate some results on data sets where the long reads were available, e.g., from Genome In A Bottle.

3.3.4 Variation Databases

Variation databases provide a way to record and share reported variations discovered in individual or population studies. The recorded events are tracked with a unique ID or accession number allowing researchers to refer to the exact variant. Some databases have linked known variants to phenotypes or diseases. These resources can help focus down analysis based on type study and provide hints to as to where it is more likely to find variations. These variation databases also help as a validating power for detected variations. Several databases are referenced below.

Database of short Genetic Variations - dbSNP

The database of short genetic variations, called dbSNP, is an archive for genetic variation across different species developed by the National Center for Biotechnology Information (NCBI) in collaboration with the National Human Genome Research Institute (NHGRI). This database not only stores a collection of SNPs, but also, short indels, short tandem-repeats, multi nucleotide polymorphisms (MNPs), and other short variants.

dnSNP is accessible at `https://www.ncbi.nlm.nih.gov/snp/` and has more than $650,000,000$ variants.

Database of Genomic Structural Variation - dbVar

The dbVar database has been developed to archive information associated with large scale genomic variation, including large insertions, deletions, translocations and inversions. In addition to archiving variation discovery, dbVar also stores associations of defined variants with phenotype information [51].

dbVar is accessible at `https://www.ncbi.nlm.nih.gov/dbvar` and provides large variants >50 bp including insertions, deletions, duplications, inversions, mobile elements, translocations, and complex variants.

Their structural variant sets are also available through github at `https://github.com/ncbi/dbvar`.

clinVar

ClinVar aims to provide a public tracked record of reported relationships between human variation and observed health status with supporting evidence. Related evidence and information is provided through hyperlinks on the records [52].

ClinVar is accessible at `https://www.ncbi.nlm.nih.gov/clinvar/`.

Database of Genomic Variants - DGV

The Database of Genomic Variants is a curated catalogue of human genomic structural variation. The objective of the database is to provide a comprehensive summary of structual variation in the human genome (SVs >50 bp) [53]. The database also incorporates a genome browser.

DGV is accessible at `http://dgv.tcag.ca/dgv/app/home`

Database of Genomic Variants - DGVa (archived)

The database of genomic variants archive (DGVa) is a repository that provides archiving, accessioning and distribution of publicly available genomic structural variants, in all species [54].

Note : The submission, archiving, and presentation of structural variation services offered by the DGVa is transitioning to the European Variation Archive (EVA , introduced below). All of the data shown in the DGVa website is already searchable and browsable from the EVA study browser. DGVa ceased to accept direct submissions at the end of 2019 and suggests that the submissions should be made to the EVA.

DGVa is accessible at `https://www.ebi.ac.uk/dgva/`

European Variation Archive - EVA

The European Variation Archive (EVA) is an open-access database of all types of genetic variation data from all species [55].

All users can download data from any study, or submit their own data. It is also possible to query all variants in the EVA by study, gene, chromosomal location or dbSNP identifier. The EVA also provides a variant browser.

It is acessible at `https://www.ebi.ac.uk/eva/`

Albeit being an all species database, as of january 2020 the structural variants (>50 bp) studies are 92.8% human. The short variant studies are more diverse, nevertheless the human species still represents 53.3% of the studies. [55]

JVar-SV

JVar-SV is the Japan variation database for structural variation and is accessible at `https://www.ddbj.nig.ac.jp/jvar-sv/index-e.html`

3.4 Existing Software

This section references software that has been developed for SV and/or CNV calling. Tables 3.1, 3.2, 3.3 list the software alphabetically. Columns include the SV types detected by the software, the detection methods used, the input data type, as well as, the last update date and references.

The software list in the tables was meant to be as complete as possible but is not necessarily exhaustive. The most complete software list was found in [56]. The structural variation in the sequencing era review [47] introduced some extra software. The other software listed were found through more general literature review, as well as on software repositories. For more details on a specific software see the reference in the table.

The **SV Type** field in the tables refers to the types of SVs detected by the software. This is the list of acronyms used :

- **CPX** Complex Rearrangements
- **CNG** Copy Number Gain
- **DEL** Deletions
- **DUP** Duplications
- **INS** Insertions
- **INV** Inversions
- **MEI** Mobile Element Insertions
- **NAHR** Non-Allelic Homologuous Recombination
- **NUMT** Nuclear Mitochondrial DNA insertions
- **TRX** Translocations
- **VEI** Viral Element Insertions

Software	SV Types	Detection Methods	Input Data Type	Last update	References
1-2-3-SV	DEL, INS, INV, TRX	RP	Short Reads	May. 2012	[57]
adVNTR	Variable number Tandem Repeats	HMM	Short + Long Reads	Dec. 2019	[58]
AS-GENSENG	DEL, DUP	RP, AS	Short Reads	Jun. 2016	[59]
BASIL-ANISE	INS	RP, AS	Short Reads	Aug. 2017	[60]
BatVI	VEI	RP, SR	FASTQ	Oct. 2018	[61]
BICseq2	DEL, DUP	RD	BAM	Nov. 2019	[62]
Bionano Solve	DEL, DUP, INS, INV, TRX	Optical Mapping Discordance	Optical Mapping Bionano	?	[63], [64], [65]
BreakDancer	DEL, INS, INV, TRX	RP	Short Reads	Nov. 2015	[66], [67]
BreakSeek	DEL, INS	RP, SR	BAM	Nov 2016	[68]
BreakSeq2	DEL, INS	SV-library	BAM	Nov 2015	[69], [70]
Breakway	DEL, INS	RP	BAM	Apr. 2013	[71]
Canvas	DEL, DUP, INS, tandem-repeats	RD	Short Reads BAM	Sep. 2019	[72, 73]
CLEVER	DEL, INS	RP	BAM	Oct. 2019	[74]
cn.MOPS	CNVs, CNAs	RD, statistical modeling	Short Reads	Oct. 2019	[75]
CNVeM	DEL	Bayesian approach	?	?	[76]
CNVer	CNVs	RP	Short Reads	?	[77]
CNVnator	DEL, DUP	RD	BAM	Dec. 2019	[78]
CNVrd2	CNVs	?	BAM	Jan. 2016	[79], [80]
CODEX/CODEX2	CNVs	RD	BAM	Jun. 2019	[81]
Control-FREEC	DEL, DUP	RD	BAM	Oct. 2019	[82]
CoNVaDING	Single Exon CNVs	RD	BAM	Aug. 2019	[83]
CORGi	Complex SVs	SR, SP on LR	Long Reads BAM	Jun. 2018	[84]
CREST	DEL, INS, INV, TRX	RP, SR, AS	BAM	Apr. 2016	[85]
DELLY	DEL, DUP, INV, TRX	RP, SR	BAM	Jan. 2020	[86]
DIGTYPER	DUP, INV	RP, SR	BAM	Dec. 2017	[87]
DINUMT	NUMT	RP	BAM	Sep. 2014	[88]
ERDS	DEL, DUP	RP, SR, RD	BAM	Jan. 2020	[89]
FermiKit	DEL, INS	AS	FASTQ	Nov. 2017	[90]
forestSV	DEL, DUP	RP, SR, RD	BAM	-	[91]
FusorSV	Dependent on input truth set	Combination of 8 callers trained on input truth set	Results from other callers and truth set	Oct. 2019	[92]
GASVpro	DEL, INV	RP, RD	BAM	Aug. 2014	[93]
GATK CNV	DEL, DUP	RD	BAM	Aug. 2019	[13]
gemtools	Complex SVs, SV Phasing	RP (linked reads)	10x Genomics BAM, VCF	Jul. 2019	[94]
GenomeSTRiP	DEL	RP, RD	BAM	2016	[95], [96]
GRIDSS	DEL, DUP, INS, INV, TRX	RP, SR, AS	BAM	Dec. 2019	[97]
GROC-SVs	Complex SVs from any type	AS	10x Genomics BAM	Oct. 2019	[98]
GROM-RD	CNVs	RD	BAM	? 2015	[99]
HGT-ID	VEI	RP, SR	BAM	? 2018	[100]
HugeSeq	DEL, DUP, INS, INV	Ensemble	FASTQ	Jul. 2015	[101]
Hydra-sv	DEL	RP, AS	BAM	Oct. 2014	[102]
HySA	DEL, INS, CPX	AS	Short Reads + Long Reads (hybrid)	Sep. 2019	[103]
iCopyDAV	DEL, DUP	RD	BAM	Mar. 2018	[104]
iSVP	DEL	Ensemble	Short Reads BAM	?	[105]
indelMINER	DEL	RP, SR	BAM	? 2015	[106]
inGAP-sv	DEL, DUP, INS, INV	RP, RD	SAM	? 2010	[107]
ITIS	MEI	RP, SR	BAM	? 2015	[108]
laSV	DEL, DUP, INV, TRX	AS	FASTQ	Sep 2016	[109]
LinkedSV	DEL, DUP, INV, TRX	RP (Linked-reads)	10x genomics	Nov. 2019	[110]
Local Rearrangements	Complex SVs	LR	Multiple Alignment Format .maf (with LAST http://last.cbrc.jp)	Nov. 2017	[111]

Table 3.1 List of SV detection software

Software	SV Types	Detection Methods	Input Data Type	Last update	References
Long Ranger	DEL, DUP, INV, TRX	RP (Linked-reads)	10x Genomics	Jul. 2018	[112]
Lumpy	DEL, DUP, INV, TRX	RP, SR, RD, Meta	BAM, calls from other software	Apr. 2019	[48]
Manta	DEL, DUP, INS, INV, TRX	RP, SR, AS	BAM	Jul. 2019	[113], [114]
MATCHCLIP(s)	DEL, DUP	RP, SR, RD	BAM	Feb. 2014	[115]
Meerkat	DEL, DUP, INS, INV, TRX	RP, SR	BAM	? 2015	[116]
MELT	MEI, (NUMT)	RP, SR	BAM	Nov. 2019	[117]
MetaSV	DEL, DUP, INS, INV, TRX	Meta	BAM Results from other callers	Jan. 2017	[49]
MindTheGap	INS	k-mer AS	FASTQ	Nov. 2019	[118]
Mobster	MEI, (NUMT, VEI)	RP, SR	BAM	Apr. 2014	[119]
MrCaNaVaR	Gene copy number estimates	RD	SAM	Sep. 2013	[120]
MultiBreak-SV	DEL, INV, TRX	Ensemble	Short reads + Long reads (hybrid)	Jul. 2019	[121]
NAIBR	DEL, DUP, INS, INV, TRX	RP (Linked-reads and short-reads)	10x Genomics, Illumina	Oct. 2019	[122]
NanoSV	DEL, DUP, INS, INV, TRX	SR, SP our LR	Oxford Nanopore	Apr. 2019	[123]
NextSV	DEL, DUP, INS, INV, CPX, TRX	Meta	Results from other callers (PBhoney and Sniffles)	Jan. 2020	[124]
Novel-X	INS	RP (Linked-reads) + Unmapped AS	10x Genomics	Sep. 2019	[125]
nplnv	NAHR-mediated Inversions	Multiple Align LR	Long Reads	Aug. 2019	[126]
OMSV	DEL, DUP, INS, INV, TRX	Optical Mapping Discordance	Optical Mapping Bionano	Mar. 2019	[127]
OncoSNP-SEQ	DEL, DUP	RD	BAM	Jun 2014	[128]
PacmonsTR	Tandem Repeats	LR	Long Reads PacBio	Sep. 2015	[129]
PALMER	MEIs	LR	Long Reads	Dec. 2019	[130]
Pamir	INS	RP, SR, AS	BAM	Jan. 2020	[131]
Parliament2	DEL, DUP, INS, INV, TRX	Meta (6 callers) Trained Matrix (HG002)	Results of other callers	Jul. 2019	[132]
PBHoney	DEL, INS, INV, TRX	SR on LR	Long Reads PacBio	2017 ?	[133]
pbsv	DEL, DUP, INS, INV, TRX	SR on LR	Long Reads PacBio BAM	Aug. 2019	[134]
PennCNV	DEL, DUP	RD	BAM	Dec. 2019	[135]
Picky	DEL, DUP, INS, INV, TRX	SR, SP on LR	Long Reads Oxford Nanopore	Jul. 2018	[136]
Pindel	DEL, DUP, INS, INV, TRX	SR	BAM	May 2017	[137]
PopIns	INS	RP, SR, AS	BAM	Jan. 2020	[138]
PRISM	DEL, INS, INV	RP, SR	BAM	Dec. 2013	[139]
RAPTR-SV	DEL, DUP, INS	RP, SV	BAM	Mar. 2016	[140]
readDepth	DEL, DUP	RD	BAM	Jul. 2016	[141]
RepeatHMM	Microsatellites	HMM, LR	Long Reads	Jun. 2019	[142]
RetroSeq	MEI	RP, SR	BAM	May. 2017	[143]
rMETL	MEI	LR	Long Reads .sam, FASTA	Oct. 2019	[144]
SDA	Segmental Dups	LR	Long Reads .bam	Jan. 2020	[145]
SMRT-SV	DEL, DUP, INS, INV	Local AS on LR	Long Reads	Feb. 2019	[146]
SMRT-SV 2	DEL, DUP, INS, INV	Local AS on LR	Long Reads	Dec. 2019	[147]

Table 3.2 List of SV detection software

Software	SV Types	Detection Methods	Input Data Type	Last update	References
Sniffles	DEL, DUP, INS, INV, CPX, TRX	SR, SP on LR	Long Reads	Dec. 2019	[148]
Socrates	DEL, INS	SR	BAM	? 2014	[149]
SoftSearch	DEL, DUP, INS, INV, TRX	RP, SR	BAM	Jul. 2016	[150]
SoftSV	DEL, DUP, INV, TRX	RP, SR	BAM	Sep. 2015	[151]
SoloDel	DEL	RP, RD	BAM	Mar. 2015	[152]
SpeedSeq	DEL, DUP, INS, INV, TRX, CNG	Ensemble Bayesian	Short Reads BAM	Feb. 2018	[153]
Sprites	DEL	SR	BAM	Jul. 2016	[154]
SurVIndel	DEL, DUP	RP, SR, statistical model	BAM	Sep. 2019	[155]
SV2	DEL, DUP	RP, SR, RD	BAM	Dec. 2019	[156]
SvABA	DEL, DUP, INS, INV	RP, SR, AS	BAM	Sep. 2019	[157]
SVDetect	DEL, DUP, INS, INV, TRX	RP	BAM	Jan. 2013	[158]
SVelter	DEL, DUP, INV	RP, SR, RD	BAM	Nov. 2018	[159]
SVfinder	DEL, INS, INV, TRX	RP	BAM	Feb. 2014	[160]
SVIM	Complex SVs	SR, SP on LR	Long Reads PacBio	Jan. 2020	[161]
SVMerge	DEL, INS, INV, CNG, CPX	Ensemble	BAM and other SV callers results	Aug. 2012	[162]
SVSeq2	DEL, INS	SR	BAM	Nov. 2014	[163]
tandem-genotype	Tandem repeats	LR	Multiple Alignment Format .maf (with LAST http://last.cbrc.jp)	Jun. 2019	[164]
Tangram	MEI	RP, SR	BAM	Feb. 2015	[165]
Tea	MEI	RP, SR	BAM	Jun. 2019	[166]
TEMP	MEI	RP, SR	BAM	Jul. 2017	[167]
TIDDIT	DEL, DUP, INV, TRX	RP, SR, RD	BAM	Jan. 2020	[168]
TSD	Complex SVs	LR	Long Reads, e.g., PacBio FASTQ	Nov. 2019	[169]
Ulysses	DEL, DUP, INV	RP	BAM	Sep. 2015	[170]
VALOR2	Segmental Dups	RP (Linked Reads)	10x Genomics bam	Dec. 2019	[171]
VariationHunter (Now TARDIS)	DEL, INS, INV	RP	BAM	Jan. 2020	[172], [173]
VirusFinder2	VEI	Others	BAM	Jun. 2014	[174]
VirusSeq	VEI	RP	BAM	? 2013	[175]
Wham	DEL, DUP, INS, INV	RP, SR	BAM	May 2017	[176]
XHMM	CNVs	PCA & HMM on RD	BAM	Jan. 2020	[177]
ZoomX	DEL, DUP, INV, TRX	RD (linked molecule coverage)	10x Genomics	Sep. 2019	[178]

Table 3.3 List of SV detection software

The **Detection Methods** refer to the algorithms used for detection as classified above and sometimes give extra information when known, for the exact details please refer to the reference.

The acronyms used stand for :

- **AS** Assembly based approach
- **LR** Long Reads
- **SP** Signals within the span of reads
- **SR** Split Reads (Clipped Reads)

- **RD** Read Depth (Coverage)
- **RP** Read Pairs (e.g., fragment length or pair orientation)

The **Input Data Type** field refers to the type of data used by the software, in most cases the input is composed of short reads (e.g., Illumina paired-end short reads) or long reads (e.g., PacBio, ONT). Some software take other inputs such as optical mapping data or even data generated from other software (e.g., variant calls). When only the format is specified (e.g., BAM of FASTQ) it means that it accepts these input formats and that the exact input is not necessarily dependent on a single sequencing technology. For example, a read depth approach could be done with any type of aligned reads (e.g., short, long, of any type of sequencing). For more details refer to the references.

The **Last update** field refers to the last update to the software or documentation as of January 2020. This field was filled by retrieving the last commit date on the main (master) branch of the software repository when available. Else it can be the most recent update to the documentation or software website. This field gives an idea of the activity of the software or its development. In order to get the exact state of the software the reference should be checked. The reference not only cites the paper but also gives an *URL* to the software repository or website directly. Fields marked with "?" are approximate or unknown.

The **References** cite the publication (if there is one) for the software and include an *URL* to the web resource of the software as explained above. This is either the source code repository or website where the software can be downloaded.

3.4.1 Comments on existing software

The sheer number of software callers developed over the years is a sign that there is no clear solution on how to solve the problem of SV and CNV detection. There is no "silver bullet". This is also why meta algorithms and frameworks combining existing callers have appeared.

A comprehensive benchmark of 69 SV detection software can be found in [56]. Other recent studies can be found in [179], [180] and [181]. Most publications of software include benchmarks comparing the software to some of the existing solutions, this can also serve as a comparison. In order to assess the usefulness or efficiency of a particular software it should be benchmarked in a relevant setting depending on the input data type and project. The compute power and time available can be a consideration as well. Finally the target type of variations is probably the most important aspect. It should also be noted that the software often is optimized with a particular goal in mind, such as for example, somatic cell SV detection. It is also important to check whether the software was created with whole genome sequencing data in mind or only whole exome sequencing data.

With the list of software provided in the tables 3.1, 3.2, and 3.3 the intention was to provide references for the reader to check out and solutions for the researcher to try out. The best software based on the research of Kosugi et al. [56] will also serve as the basis for the evaluation of calling performance in the results chapter.

3.5 Summary and Goal of this Project

The signs of structural variation on short-read sequencing aligned reads was analyzed in this chapter and this analysis gives hints as to how to use the alignment data to discover and assess CNVs. Databases of known variations and the current state of the art in structural variation software were given as a reference. The analysis will guide the development of strategies and algorithms presented in the next chapter.

For this project it was proposed to develop a software tool that would call CNVs from a whole genome sequencing input alignment file, the main goals where the following : Being able to predict and identify breakpoints of structural variation, identify CNVs regions as precisely as possible in terms of positions of the start and stop breakpoints of the events (and therefore also the event size). The goal is also to do this analysis in a timely manner for whole genome sequencing datasets (e.g., 30x coverage alignment files).

This will take the form of a software framework or collection of tools that can take a BAM alignment file as input and generate one or more Variant Calling Format (VCF) files as output. A white-box method is preferred as to be able to understand and analyze the detection methods and results in contrast to, for example, deep learning trained models where the impact of the topology and weights could be hard to interpret given the results.

4 | Strategies and Algorithms

Contents

4.1 Introduction

Many of the algorithms presented in this section are generic or abstract by design, this is to make it possible and easier to change them based on the specific needs of each task in which they will be used. Also, because the implementation was an iterative process, it was better to work with generic algorithms. They also provide better flexibility to create a multi-purpose CNV assessment framework and can be tuned for specific usages when combined with benchmarks.

This chapter will go over algorithms to extract signals and features from the alignment data, prediction algorithms for regions of breakpoints, calling and solving algorithms, and finally assembly algorithms. These are the main tools and strategies used for this project.

4.2 Signal algorithms

Signals are defined here as a value relative to a position in the genome. $S = f(x)$ where x is a position in the genome. They should be defined for every x in the genome, If the source of a given signal is not defined for every x we can simply assume a default value for every undefined position.

Signals can be created from any source. However, they will always be relative to the reference genome used. So more formally a signal is a function $S = f(x)$ where $f(x)$ is defined for every position x in the reference genome. x will be in the form $< c, p >$ where c is the contig (e.g., chromosome or decoy) of the reference and p is the position inside that chromosome.

4.2.1 Signal Extraction from Aligned Reads

Values $f(x)$ over the reference genome positions x for the aligned reads (figure 4.1) are created in the following way. The reference is split into sub-regions (figure 4.2) in order to apply a divide and conquer strategy allowing for parallel computation of signals on sub-regions as well as reducing the necessary memory footprint.

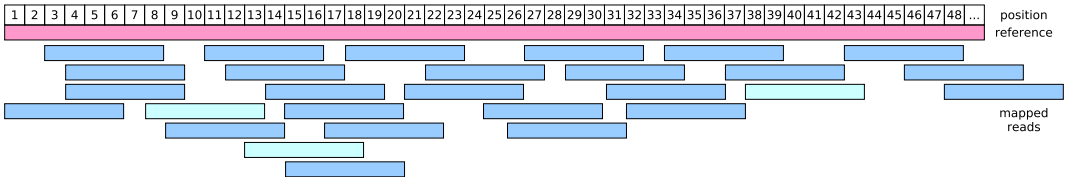


Figure 4.1 Aligned reads to a reference

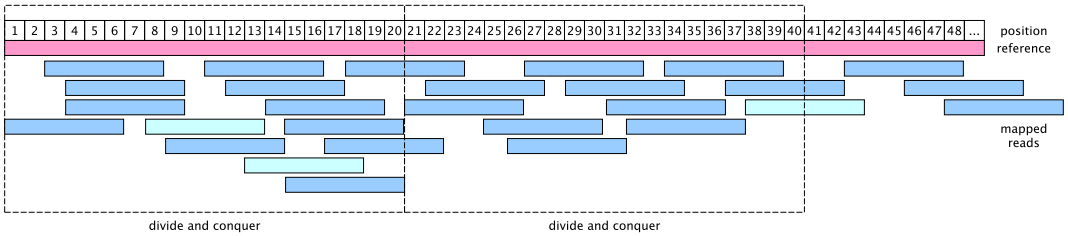


Figure 4.2 Regions are split for individual processing

- First, the aligned reads are filtered given a predicate (figure 4.3, and 4.4), e.g., remove reads that are of bad mapping quality, or remove reads that are mapped to multiple locations.

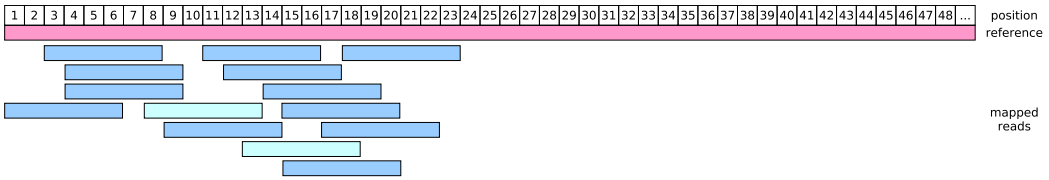


Figure 4.3 Reads overlapping the first split of the region

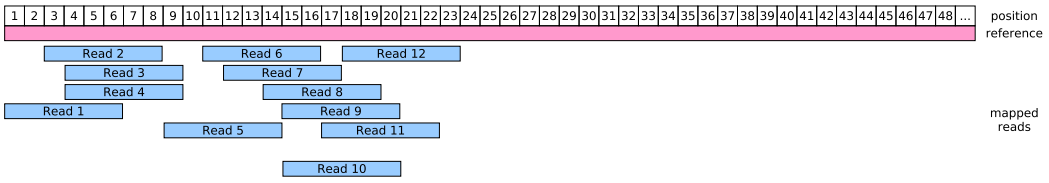


Figure 4.4 Marked reads (reads that satisfy a given predicate) have been filtered out

- Second, for all the reads, get their starting position and ending position, for each starting position assign an increment depending on a function that extracts a value from the read properties, this could be a simple constant "1", and for each ending position decrement by the same amount[1] (figure 4.5).

  Having the increment and decrement function be arbitrary allows to generate more complex signals such as the sum of the edit distance between the reads at a given position and the reference, this also allows for an asymmetric scoring between increments and decrements based on independent parameters.

[1] For signal generation using a single value representing the variation for the position would suffice, however this looses the information by how much the increments and decrements contribute to the overall variation of the signal at this location. The two are kept in order to be able to extract the increments and decrements separately.

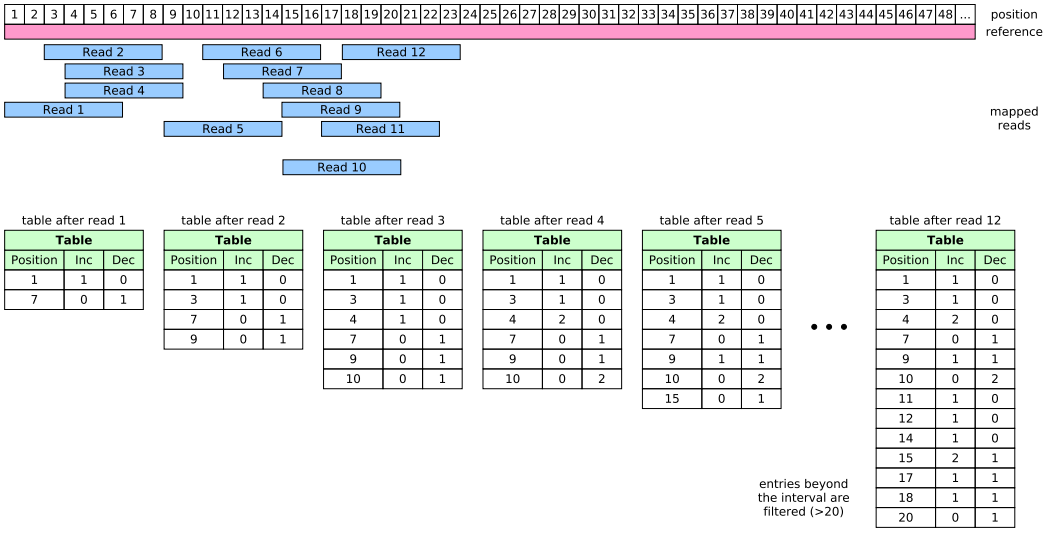


Figure 4.5 Reads are traversed to generate the increment & decrement table

- Finally, once all the increments and decrements for all the starting and stopping positions have been generated, the table is traversed and the signal is generated for the current sub-region (figure 4.6).

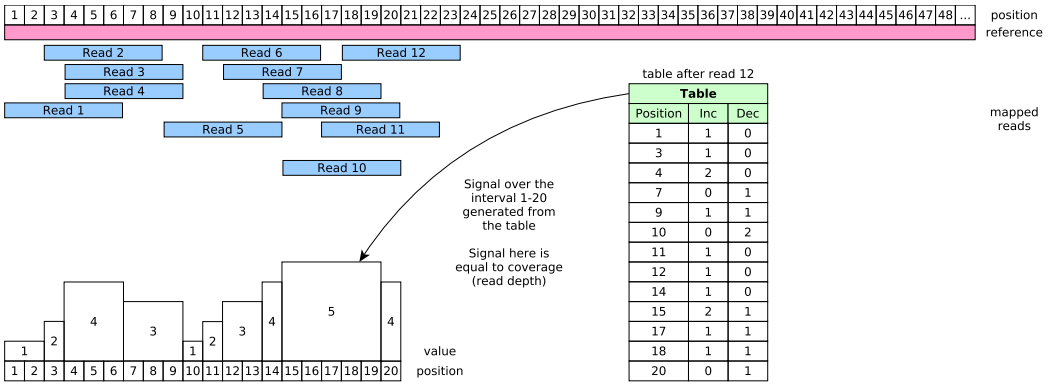


Figure 4.6 The table is transformed to the actual signal

These operations are repeated for the next sub-region as can be seen in figure 4.7. Reads overlapping the sub-region must also be taken into account in order to generate the signal correctly.

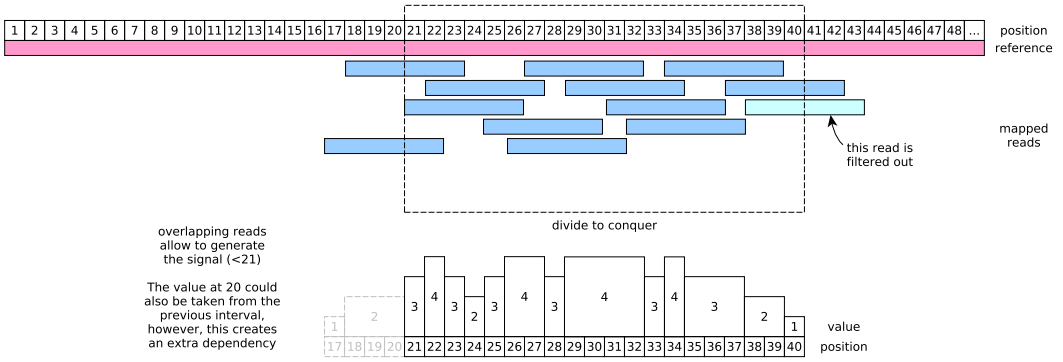


Figure 4.7 The operations are reiterated on the next sub-region

This will end up in a signal f(x) where x is a position in the genome. The signals are encoded as valued intervals, like in the BED/bedgraph format giving a first level of compression. For practical aspects theses signals are then saved into gzip compressed bedgraph files, this also allows them to be loaded into IGV rapidly.

Because the accumulated increments of the starts and ends of the reads are stored inside a table (hash-table), the table will grow bigger and bigger when traversing more and more reads in a region increasing its memory footprint. Look-ups in a bigger table can become more costly as well. Therefore the signal is generated over smaller sub-regions of the genome and extracted from smaller tables before being written to the output file. The caveats of this methods are, first, that the reads overlapping the end and start of a smaller interval will be processed twice and, second, that the output encoded file may have two intervals of the same value being split. This split is however not a problem for analysis and could be merged easily if needed. For this project the human chromosomes were split into a thousand contiguous sub-regions (value chosen arbitrarily), this is a parameter of the algorithm and may be tuned in the future to give better performance.

Examples of signals that can be extracted with this algorithm are numerous, here are a few : coverage, number of read-pairs with unexpected fragment length when aligned, number of reads that have small indels in the alignment, sum of the edit distance between the reads and reference for a given region, number of pairs that have a discordant orientation, number of clipped reads, etc. Chapter 7 will show concrete examples of extracted signals.

Thanks to the fact that this algorithm takes an arbitrary function that extracts increment and decrement scores from an aligned read, this algorithm can be used to extract any property from the reads. Utilizing a function that always returns $(1, -1)$ will give coverage. Utilizing a function that returns $(1, -1)$ when the read is clipped else $(0, 0)$ will give the number of clipped reads. Utilizing a function that returns the edit distance ed as $(ed, -ed)$ for the increment and decrement score will give a signals that is the sum of edits of aligned reads for a given region, etc. The last example shows that the signal is not only proportional to the number of reads that satisfy a property but also to the property itself (edit distance in that example). Thanks to the fact that our reads have mates and form pairs, the extracted signals can also be function of

this relationship. For example, return $(1, -1)$ when both reads in the pair have the same orientation else $(0, 0)$, this will give the number of reads that form a pair with an unexpected orientation (this can happen with inversions).

4.2.2 Signals from other sources

Some signals are interesting for analysis but do not necessarily come from the aligned reads. For example, a "mappability" signal which encodes if a region of the genome is easily mapped by reads of a given size. Highly repetitive regions or regions with low complexity will be difficult to map to. This kind of signal could be used to filter some of the results. Another example could be a signal that encodes regions of lesser stability over the genome. There are many possibilities here so it is important to keep an input for other signals.

The idea is to write parsing algorithms for files such as VCF, BED, BedGraph, Tab delimited formats (tabix), CSV, etc. and adapters functions that transform the input data in the valued interval format. Parsing algorithms are defined to take an input file and generate a collection of entries (specific to the input file type) and adapters are function that can be mapped onto these collections to generate an output collection of the desired type (valued interval format). This can be done directly or as of a "view" meaning that the transformation is not applied unless the data is queried.

4.2.3 Signal encoding

The signals generated above are encoded as a contiguous region (interval) on the genome or as denoted above "valued intervals". This means encoded by the start and stop position on a given contig (chromosome) and a value. This encoding regroups contiguous regions with the same value as would be the case with run length encoding (RLE). This encoding has also the advantage over RLE that access to a given position can be done faster (as with a dictionary search).

An example is given in table 4.1 to clarify some details about the encoding. The stop position is non inclusive, therefore a value for a single base will be $[\text{start}, \text{start} + 1[= [\text{start}, \text{stop}[$ and for a eight base interval $[\text{start}, \text{start} + 8[= [\text{start}, \text{stop}[$ as shown in the table.

Contig	Start	Stop	Value
chr8	43834647	43834648	15
chr8	43834648	43834656	42

Table 4.1 Valued Interval Encoding of a signal example

The values of the signal can be of any type, an integer value, a floating point value, or even a string value, with the restriction that for a given signal all the values are of the same type. The contig is the exact name of the contig in the reference as a string, depending on the reference used it may be in the form of "chrN" or "N". Since all signals are extracted from reads aligned to a given reference, the names of the contigs

of that particular reference are used. These contig names can be easily changed by mapping an adapter over the whole collection, this is useful because different versions of the references can use different names for the chromosomes.

Note : This encoding is the same used in the BED or Bedgraph format and can therefore be directly stored with one of these formats.

4.2.4 Signal processing

It is interesting to apply some signal processing to the signals extracted from the aligned reads. In order to assess copy numbers of a gene it could be interesting to do a low pass filter on the read depth signal and estimate the gene or loci repeat number based on the coverage given by this filtered signal.

Another useful processing algorithm exampls is edge detection, this is useful to assess breakpoints around events such as deletions or duplications. For example, it is expected to see a substantial drop in coverage at a deletion site. The edges can be found by analyzing the read depth signal through an edge detection filter.

Edge detection algorithm

The edge detection algorithm used in this project is given by equation (4.1)

$$Edge_x = \begin{cases} \text{edge} & \text{if } f(x+1) - f(x) > threshold \\ \text{no edge} & \text{otherwise} \end{cases} \tag{4.1}$$

This corresponds to a threshold on the derivative of the signal $\frac{\Delta f(x)}{\Delta x} = \frac{f(x+h)-f(x)}{(x+h)-(x)} = \frac{f(x+1)-f(x)}{(x+1)-(x)} = f(x+1) - f(x)$ where h is the smallest step in our discrete signal. Or in signal processing terms, applying the convolution kernel $[-1, 1]$ followed by a thresholding step.

This is a very cost effective and simple processing step, requiring only a single subtraction for every position and a filtering (thresholding) step. There is another benefit, because the data is interval encoded and due to encoding for any given interval the value of the signal is the same. Therefore, Δf will be 0 meaning, it is only necessary to compute the Δf when a new interval is encountered. This can significantly reduce the number of operations needed to compute the result. In the worst case, where all the intervals are of unit length, the processing will be the same as with a traditional signal and a computation must be applied for every position. On the other side, for signals that have large intervals with the same value a lot less of operations are required to do the computation.

Filtering

Low pass filtering, e.g., averaging, can be useful to "smooth" out the signals and may be useful for applications such as global estimation of the copy number over larger regions.

High pass filtering is useful to find transients in the signal as is done above with the derivation step of the edge detection algorithm.

Any signal processing filters can be applied on the signals if they represent numerical values, it is however either required to change the algorithms to be compatible with valued interval encoding or transform the encoded signals back to the standard form (sequence of values). This can be done by creating an iterator representing the signal that will return the value of the interval for each step inside the interval and will get the value from the encoded signal for the next interval when needed.

If the signals are interpreted as a set of regions, they can also be filtered from a set theory point of view. For example the signal could be filtered by removing all subregions outside a given span. Or remove all entries, all intervals, that satisfy a given predicate, e.g., on their value.

Interpretation of signals

Being able to interpret signals as sets makes this encoding very versatile. Even when the signal has been filtered from a set theory point of view and is now missing several entries (i.e., being undefined for some positions, having gaps) it can be transformed back into a complete signal defined for every position quickly be returning a default value for undefined positions. This value could be zero or a more abstract value such as NaN (not a number). Using NaN can be helpful because it will return NaN in computations where a value from an undefined region is used making it easy to filter the resulting signal again. The filtered signal will only have the regions that have a defined value. And this was possible without any modification to the signal processing algorithm itself.

Being able to represent the extracted signals in two ways, as a continuous signal $f(x)$ for a position x or as a set A with valued sub-regions (intervals) $r \in A$ where $r = <\text{contig}, [\text{start}, \text{stop}[>$ allows to seamlessly apply signal processing algorithms as well as set theory algorithms. Giving a wide variety of linear and non-linear operations available for experimentation.

The specific way the signals are processed and viewed as is really dependent on the application, examples will be given below for signals used in prediction and calling algorithms.

4.3 Prediction algorithms

This section introduces the generic algorithms used to predict regions of interest and breakpoints, which are locations where variation starts or ends compared to the reference. These algorithms are generally simple extraction of features from the signals generated with the algorithms and strategies above.

4.3.1 Breakpoint prediction

Breakpoint prediction takes one or more signals as inputs and outputs positions were variation events started or stopped relative to the reference genome. The resolution of these positions is not necessarily a per base resolution but can also be an interval around a position. This is because the start and stop positions can be "fuzzy" or even undefined.

With breakpoint position prediction the main focus is often a position where there is an unexpected variation in signals, for example, a massive difference in the read depth. E.g., in case of a deletion the read depth is expected to drop more drastically around the breakpoint. Another example could be the presence of a substantial amount of clipped reads relative to the total of locally mapped reads.

Breakpoints can also be inferred from read characteristics such as read pairs with incoherent read direction and read pairs where each read maps to another chromosome. In order to predict these breakpoints a simple thresholding over the correct signal is often enough. For example, if for a position more than 10 reads have their mate mapping to another chromosome the position should be called a breakpoint.

4.3.2 Region prediction

A region is a contiguous interval on a reference contig. Predicting regions that encompass structural variation can be done based on several signals. The simplest algorithm to predict a region may simply be the thresholding of a signal. For example if there is no coverage (read depth of 0) of a region, it may be predicted as a deletion. More complex regions may be called by more involved algorithms.

Extension prediction

Prediction with extension is based on finding breakpoints that open a region (start), trying to extend them based on the properties of another signal and ending the region with another breakpoint (end).

This is implemented with the following algorithm :

Algorithm 1 Extension prediction

Require: StartBreakpoints $\neq \emptyset$, EndBreakpoints $\neq \emptyset$, Intervals $\neq \emptyset$
  for all $s \in$ StartBreakpoints **do**
    for all $i \in$ Intervals such that $s \in i$ **do**
      for all $e \in$ EndBreakpoints such that $e \in i$ **do**
        Predictions $\leftarrow$ Predictions $\cup \{[s, e]\}$
      end for
    end for
  end for
  return Predictions

This algorithm may generate overlapping predictions, which based on the type of predictions may be what is needed. Algorithm 1 also considers all possible ending breakpoints in the intervals. An alternative algorithm is shown by algorithm 2 were the predictions are ended by the "best" ending breakpoint for the event. The notion of "best" or "better" is abstract and dependent on the specific predictions we want to make, it is therefore passed as an input function to the algorithm.

Algorithm 2 Extension prediction 2

Require: StartBreakpoints $\neq \emptyset$, EndBreakpoints $\neq \emptyset$, Intervals $\neq \emptyset$
  for all $s \in$ StartBreakpoints **do**
    for all $i \in$ Intervals such that $s \in i$ **do**
      for all $e \in$ EndBreakpoints such that $e \in i$ **do**
        if ϵ is undefined or e is better than ϵ **then**
          $\epsilon \leftarrow e$
        end if
      end for
      Predictions $\leftarrow$ Predictions $\cup \{[s, \epsilon]\}$
    end for
  end for
  return Predictions

The algorithm can also be turned around as seen in algorithm 3 in order to not only choose the best ending breakpoint but to chose the best pair of starting and ending breakpoints for the given interval. The notion of "better" is still abstract and a function is passed to the algorithm to determine if a pair of starting and ending breakpoints are better than another. This function can be as simple as the greatest distance between the breakpoints, yielding bigger predicted intervals. This version of the algorithm is very useful when the input intervals are regions were it is suspected to encounter variation and a set of possible breakpoints is provided, it will try to generate the best predictions for each interval.

Algorithm 3 Extension prediction 3

Require: StartBreakpoints $\neq \emptyset$, EndBreakpoints $\neq \emptyset$, Intervals $\neq \emptyset$
  for all $i \in$ Intervals **do**
    for all $s \in$ StartBreakpoints such that $s \in i$ **do**
      for all $e \in$ EndBreakpoints such that $e \in i$ **do**
        if the pair $[\varsigma, \epsilon]$ is undefined **then**
          $[\varsigma, \epsilon] \leftarrow [s, e]$
        else
          if the pair $[s, e]$ is better than $[\varsigma, \epsilon]$ **then**
            $[\varsigma, \epsilon] \leftarrow [s, e]$
          end if
        end if
      end for
    end for
    Predictions $\leftarrow$ Predictions $\cup \{[\varsigma, \epsilon]\}$
  end for
  return Predictions

For generality this algorithm can be changed to collect every predicted region and then filtered which is the same as passing a function which tells the algorithm if a given pair of start and end breakpoints should be collected as a prediction.

An example of such predictor for deletions could be implemented by taking the edges in the coverage signal as breakpoints, falling edge and rising edge as starting and ending breakpoints respectively. A property such as a coverage lower than a threshold could then be used for the intervals that join the breakpoints.

Regression trained based prediction

The predictions above are rather simple, they are meant to be, in order to be computationally inexpensive. More complex predictors can be used in order to do more effective predictions. There is always a trade-off between the complexity (cost) of the predictor and the quality of the predictions.

It is in our best interest to maximize the quality over the cost, nevertheless it can be interesting to have high(er) quality predictors even if the quality over cost ratio is worse. Especially when considering only small regions of the genome and not a whole genome prediction or when time (cost) is not a problem.

One possible approach is to use a regression trained based predictor. Its goal is still to output predicted regions with variation given the input signals. This could be implemented by training a regression model that would output the probability of a position to be a start or end breakpoint given the signals in a neighborhood (around) the position. And finally start and end breakpoints would be merged into predicted regions.

Training this model would require a truth set of known variations and their effect on the signals. This can be achieved by collecting known variants in various variant databases. Then extract the signals around the breakpoints of each variant, as well as, randomly selected non variant sites and train a predictor based on this data.

For this project it was chosen not to do this for several reasons. First, it would require to generate all the training sets and do the training which can be time consuming. Second, it will probably be very biased because of the population bias of the variant studies (e.g., some ethnicities may be more studied or represented than others). Third, running the trained model to do the predictions will still be costly because of the multiple floating point operations. Fourth, the model (trained prediction matrix) will be a black box and results may be difficult to understand and it will become hard to find out why the models mispredicted some regions or breakpoints.

Such a predictor still is a valid and interesting approach, therefore it was also chosen in this project to provide the extracted signals in order to facilitate the addition of such predictors in the future.

Machine Learning based prediction

Machine learning predictors include regression trained predictors as discussed above but expand on the topology of the possible predictors. It would be possible to use models as simple as the perceptron to more involved (deep) neural networks, convolutional neural networks or any other topology such as recurrent neural networks to detect and classify possible variation sites based on the input reads or extracted signals.

For similar reasons as above it was chosen not to consider them in this project, mainly because of their black-box nature and potentially high training and running costs. It may still be very interesting to train machine learning models to predict CNVs and SVs based on the signals generated within this project.

Machine learning trained models may be interesting to classify detected variations instead of predicting them. However as will be seen below in section 4.4 there are some (very) simple methods to classify variations based on the sequencing reads which are entirely based on white-box approaches and therefore generate results that can be easily explained, in both correct and erroneous classification cases, which is not often the case with more complex classifiers.

4.4 Calling and solving algorithms

The calling algorithms take the predicted regions and based on the predictor the region may already be called as a specific variation type or only be classified as element of a set of possible variations. For example, predictions based on inferred fragment length (distance between mapped reads in a pair of reads) usually are deletions or translocations (as well as inversions depending on if the pairs were filtered based on their orientation).

So the main strategy for calling is to take into consideration the predictor that predicted the region if available, analyze the reads of the region and the relevant signals. Based on these clues the algorithm must be able to call the variation or give assumptions

about the variation. Some variations cannot be solved with short reads alone, therefore the calling algorithm should have some room to accept other type of sequencing reads such as long reads or contigs generated by de-novo assembly.

For example, novel insertions longer than the read length are difficult and sometimes impossible to call. Giving the exact sequence of the individual genome for the variant locus can be a very involved task and may require re-sequencing or using alternative sequencing technologies. The goal of this project is mainly to predict the regions with variation as precisely as possible (position of the breakpoints, i.e., start end length of the event). Calling the event as a specific subtype of variation is a nice feature to have and generating the exact sequence would be the best case.

4.4.1 Deletion calling

Deletions are maybe the simplest variations to call. If the region of the reference is not covered at all by WGS reads it probably is not there (this is region dependent and not always true). Having read pairs that span the deletion with an inferred fragment size above the expected value can also be a sign of deletion, if in this span the coverage is low or zero, we can call a deletion with confidence. If the coverage is about half in this span we can call a deletion on a single allele (loss of heterozigozity) and if the coverage is average then it is more likely to be a translocation. Cases where a region is highly different to the reference may appear as a deletion (reference region is missing) but in the individual genome there can be a novel variation of the region. This can be detected by looking at the paired reads, if they have their mate mapped elsewhere on the genome or unmapped it could be a sign of a novel variation instead of a simple deletion. The absence of unmapped mates or absence of mates mapped to other regions of the genome is therefore also a sign of real deletion.

Summary of the signs of a deletion from the analysis section :

- Drop in coverage
- Inferred fragment size too big
- Both ends of a pair of reads map around the deletion
- Deletion (small) directly visible on the aligned read
- Split reads that map to two locations (both ends of the deletion)

Calling algorithms

The main calling algorithm developed and used for this work is the following :

The signal used to determine regions of interest is the signal that counts how many reads are part of a fragment with an inferred size above the last percentile of the fragment distribution. All regions that have this signal above a given threshold are classified as regions of interest.

For every region we query the reads that have an inferred fragment size that was considered too big (above last percentile). Based on the positions of theses reads and the distribution of the positions of their mates a start and stop position is extracted for the event. If the deviation in the mate positions is too big the event is marked as a complex event else the copy number of the event is evaluated. This is done by extracting the mean coverage between the start and end of the event. If the coverage approximately is half the expected coverage we mark it as a deletion on a single allele, if the coverage is approximately zero we mark it as a deletion for both alleles, if the coverage is close or higher than the expected average coverage then the event is marked as complex (it could be a translocation or other event).

Another possible algorithm to call deletions is reporting regions with zero or very low coverage. This is a simple filter applied on the coverage signal and will expose all regions that are not covered by sequencing reads. This does not necessarily mean a deletion because the region could simply be different from the reference.

A final algorithm to call deletions is to use the signal that counts the number of elements in the alignment (CIGAR strings) of mapped reads, every-time this value is high enough there are reads that are either clipped both sides, have deletions inside the read or have insertions, the reads from those regions can then be queried to find and call the small deletions directly encoded in their alignment (CIGAR string).

4.4.2 Insertion calling

Insertions are additional bases at a given position in the genome. This can be duplications or putative sequences. The main signs of insertions are reads that are aligned correctly to the reference only up to a point of the read then highly diverge from the reference. These reads are often mapped and the end of the read that does not align well is "clipped" off, meaning it is ignored as if it was cut-off. Therefore clipped reads can be an indicator of insertions and are in general indicators of variation because they will appear where the individual genome differs from the reference.

Signs that help to call an insertion are :

- Increases in coverage (in case of duplications)
- Increases in number of clipped reads
- Clipped reads that extend outwards in both directions of a small region or even single base
- Pairs where only one read of the pair is mapped
- Insertion (small) directly visible on the alignment of the read

Calling algorithms

Calling algorithms are difficult for insertion since they often require solving a region that is not on the reference with reads that are either not mapped on the reference or incorrectly mapped. Therefore this was separated into two parts, first prediction of regions that can have insertions and second resolution via local assembly, assembly will be discussed in section 4.5. Extracting the exact sequence of the insertion is done by re-aligning the assembled sequence to the reference and will also be discussed in the assembly section below.

4.4.3 Duplication calling

Duplications are a special case of insertions, they can be called and resolved with different approaches but one is to impute the copy number from the average coverage. E.g., if the coverage of a region is 10× the expected value it may be that there are 10 copies of the region instead of the usual 2 copies (one for each allele). The exact distribution or "geometry" of the duplications may be difficult to assess, especially if the repetitions are dispersed. Local de-novo assembly on duplication events is hard because of the repetitive nature of the duplications and especially when the duplication has some mutations or novel insertions the region may not be solvable at all. Anyway, the read depth gives a good estimation of the copy number of the duplication.

Duplication calling algorithm

The algorithm used for duplication calling is an estimation of global copy number, this estimation could also be used to call other forms of CNVs such as deletions.

The global copy number signal is computed from the coverage signal with a mathematical morphology algorithm (analysis of the shape of a signal). First, the signal is segmented (separated into parts) given a predicate. Second, the signal is averaged globally for each segments (low pass filter). This gives an estimate of the average coverage for each segment. The predicate used to segment the coverage is : Segment if there is an edge (edge detection) at a given position or if we enter or leave a zero coverage region.

The estimated copy number signal generated by the method above is then filtered to keep only the regions of high copy number given a threshold, the resulting regions are merged if they are close to each other (for example within 2 read length distance). Finally the regions are filtered for a minimal length to give the final calls.

4.4.4 Inversion calling

Inversions are regions were part of the sequence has been reversed, therefore they can be called from the presence of read pairs that have both ends oriented in the same direction (forward-forward or reverse-reverse). These fragments span the breakpoints of the inversion and make it possible to identify the inversion.

More precise estimation of the breakpoints can be found by analyzing the clipped reads around the beakpoints predicted by the distribution of the reads in forward-forward or reverse-reverse pairs.

Inversion calling algorithm

The signal used to call inversions was the number of reads that have a discordant tandem orientation, if this number is above a threshold the reads are queried with their mates from the assembly file and the breakpoints are inferred from the center of the distribution of the reads on both sides.

4.4.5 Translocation calling

Translocations can be detected and called by the inferred fragment sizes, where the reads in the pair map too far away from each other, either on the same or on another chromosome.

Interchromosomal translocation calling

Breakpoints for interchromosomal translocation calling are extracted by predicting breakpoint regions, this is done by examining the signal that counts how many reads have their mate mapped to another chromosome, for regions with a high count they are marked as breakpoints and the mates are queried from the assembly. An histogram is made with the region of origin of the mates, the breakpoint as well as the distribution of the mate regions of origin is given as the final call.

Intrachromosomal translocation calling

Intrachromosomal translocations are called at the same time as deletions with the algorithm of section 4.4.1 that uses the inferred fragment sizes. If the two reads of a pair map to far from each other on the same chromosome, the coverage between them is analyzed and if this coverage is not representative of a deletion (half coverage or zero coverage) the call is marked as a translocation. Further refinement of the call set may be required to discern translocations from dispersed duplications.

4.5 Assembly algorithms

Assembly algorithms aim to recreate the original continuous sequence from which the reads come from. This is a similar problem to finding the shortest common superstring, however with assembly the goal is not necessarily the shortest common superstring but the actual string from which the reads originated. Assembly algorithms are almost always graph based and two of the main alternatives to do assembly from reads are shown in figure 4.8.

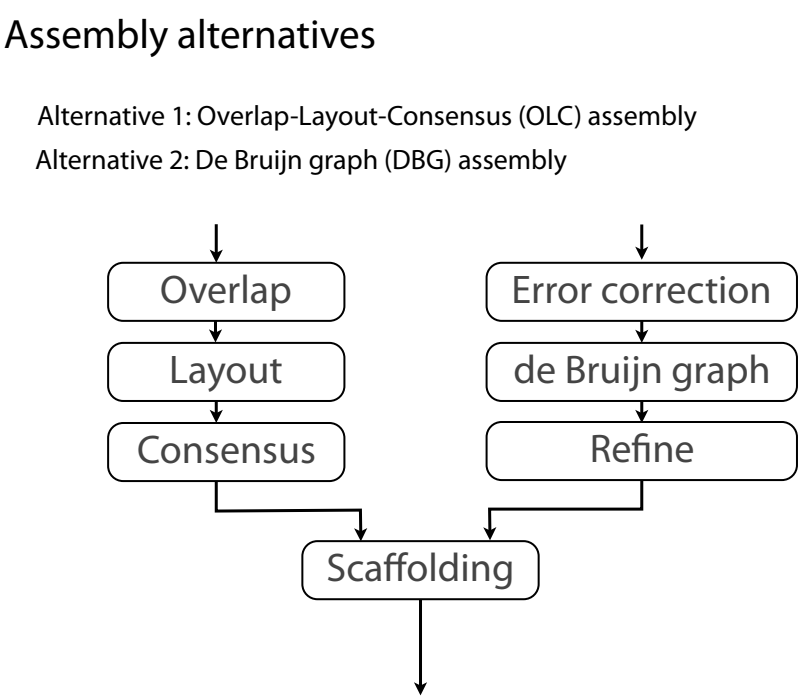


Figure 4.8 Most common assembly algorithms *Slide by Ben Langmead [182]*

The main steps of these alternatives will be presented in the three next sections (4.5.1, 4.5.2, and 4.5.3) which are based on the lectures of Ben Langmead, Department of Computer Science of Johns Hopkins Whiting School of Engineering. The original lectures are available at :
`http://www.langmead-lab.org/teaching-materials/` [2]

His lectures give an excellent overview of assembly algorithms and their specific advantages and drawbacks. The three next sections are used to give an overview of the traditional methods for assembly and will serve as a basis for the custom assembly algorithm developed for this project that is presented in section 4.5.4

4.5.1 Overlap Layout Consensus based Assembly

This method has three main steps :

[2] The lectures are published under a creative commons license `https://creativecommons.org/licenses/by-nd/2.0/` with attribution requirement and the extra requirement of signing his guestbook letting him know how his work is used.

1. **Overlap** : Build the overlap graph.
   This step consists of finding overlaps (exact or approximate overlaps) between read sequences and building a graph based on these overlaps. An example is given in figure 4.9 where the the subsequences (reads) are of length 7.

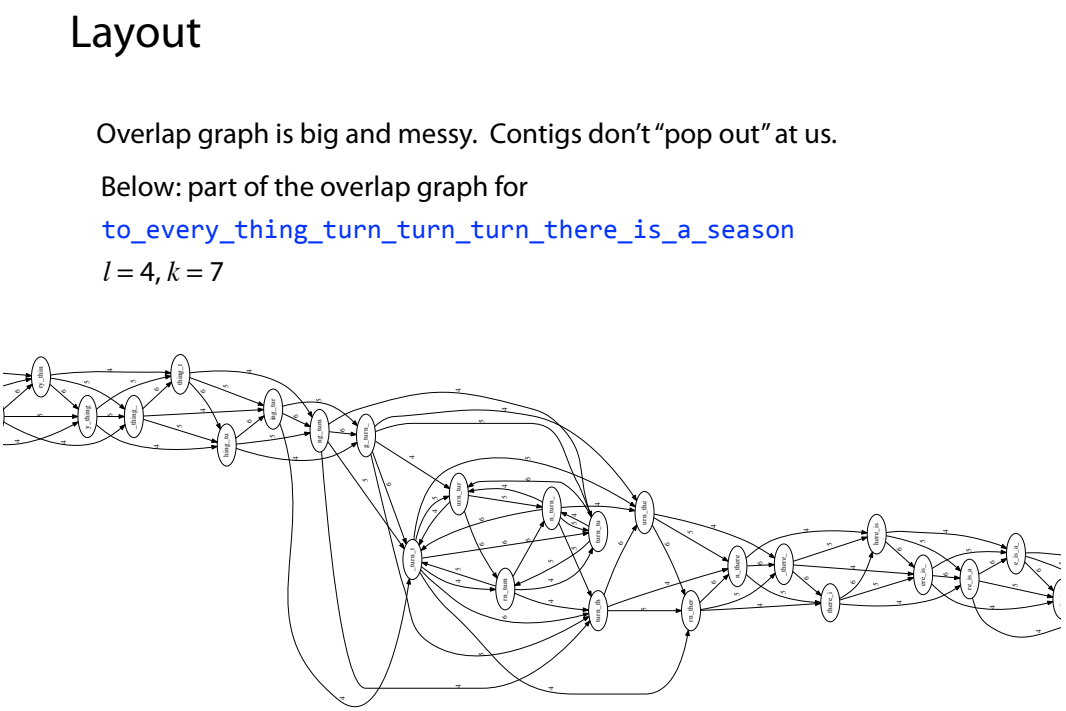
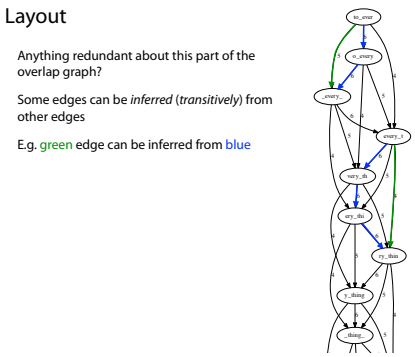
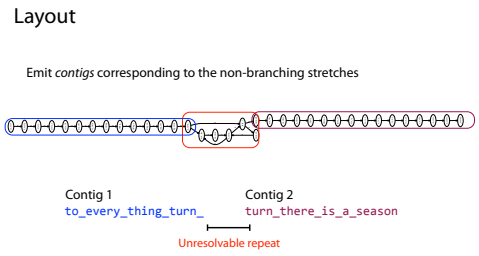


Figure 4.9 *Example OLC graph* *Slide by Ben Langmead [182]*

2. **Layout** : Coalesce paths into contigs.
   This step is done by removing transitively inferrable edges starting with edges that skip one node, then two nodes, etc. to finally emit contigs corresponding to non-branching stretches as can bee seen in figures 4.10a and 4.10b



(a) OLC layout step



(b) OLC layout end

Figure 4.10 *OLC layout steps* *Slides by Ben Langmead [182]*

3. **Consensus** : Pick the most likely nucleotide sequence for each contig.
   An example with DNA sequences is shown in figure 4.11

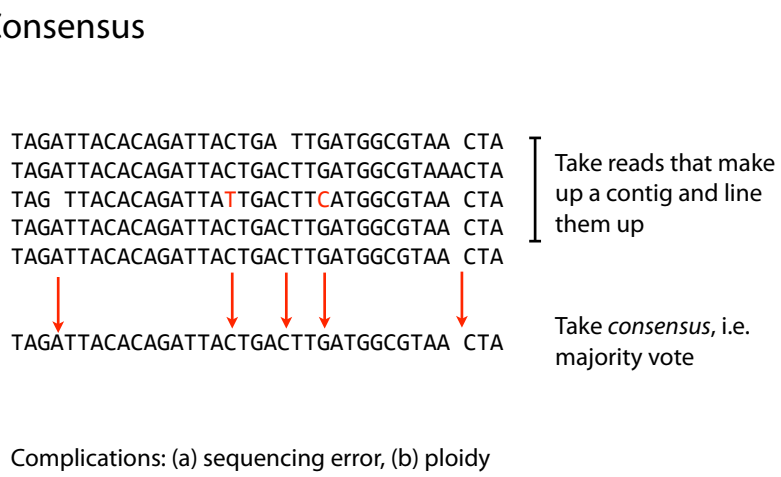


Figure 4.11 OLC consensus step *Slide by Ben Langmead [182]*

The output of the overlap consensus- layout method are assembled contig sequences, these contigs can be further assembled together with scaffolding as will be shown in section 4.5.3, or directly remapped (realigned) to the reference at this stage.

Drawbacks of the OLC method

- Building the overlap graph is slow because finding the overlaps between reads is slow (costly).
- The overlap graph is big. There is one node per read but the number of edges can grow superlinearly with the number of reads.

The details and worst case complexities are given in the lectures [182].

4.5.2 De Bruijn Graph based assembly

With the De Bruijn Graph based assembly, k-mers (subsequences of length k) are generated from the sequencing reads and each node of the graph is a k-mer that are connected by edges when they overlap by k-1 bases.

This method has three main steps :

1. **Error correction** : Remove or correct sequences that are likely to be errors.

   This step is to reduce the size of the graph because sequencing errors can generate many unique k-mers due to the low occurrence of the errors. If the reads cannot be error corrected prior to the graph construction, they can still be removed by pruning the low occurrence edges (relations that only appear once or twice are probably from errors in sequencing) and removing nodes that are not connected

after the edge pruning. Whether this is done before or after the graph creation, it will remove most of the "islands" and "tips" in the graph as can be seen in figure 4.12

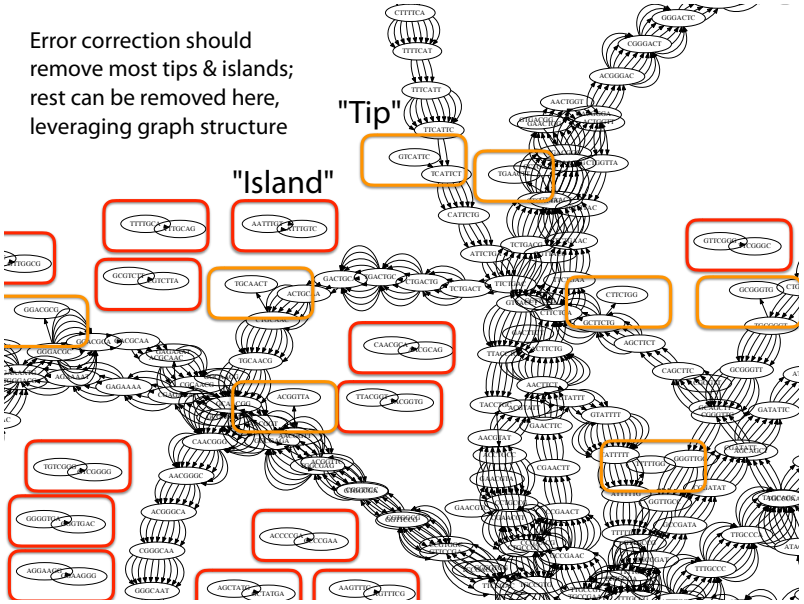
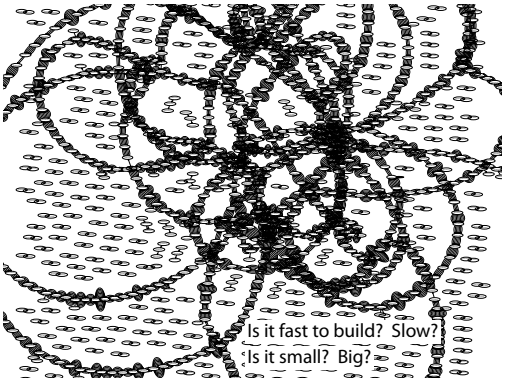


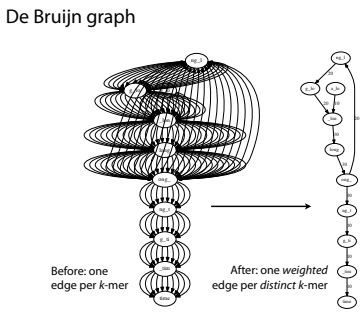
Figure 4.12 Effect of error correction on DBG graph *Slide by Ben Langmead [182]*

2. **De Bruijn Graph** : Create a graph from k-mers (subsequences of length k) where the k-mers overlap by k-1 bases.

   This is fairly straight forward because it simply implies traversing the read generating k-mers and each k-mer generated from the read is known to overlap by k-1 bases with the previous one so the overlap relation can be entered directly into the graph. Repeating overlaps can be reduced with weighted edges to make the graph more readable as can be seen in figure 4.13



(a) Unweighted DBG example



(b) DBG with weighted edges comparison

Figure 4.13 DBG examples *Slides by Ben Langmead [182]*

3. **Refine** : Simplify the graph, e.g., remove islands, trim tips, choose paths based on the reads.

Advantages of the DBG approach

- The complexity to create the graph is linear to the number of reads. The construction is very simple and requires no comparison between reads or k-mers.
- With error-free data the space required is bounded by the genome length, this is useful for high coverage because the required space will not grow linearly with the coverage because of this upper bound.
- Insertion of an overlapping k-mer pair into a De Bruijn Graph has constant complexity (if implemented with a hashtable).
- Because the overlap is of fixed size (k-1) the number of edges is bounded by the number of reads. Overlapping pairs of k-mer that appear more than once are represented by the weight of the edge.

Drawbacks of the DBG approach

- The reads are split into shorter k-mer, this makes some repeats unsolvable compared to the OLC approach.
- Only exact overlaps of k-1 bases are considered as edges, this makes the handling of sequencing errors a bit more complicated.
- This approach can lose read coherence, i.e., some paths through the graph are inconsistent with respect to the input reads. These are false paths, they exist because the same k-mer can be present in multiple reads generating paths that jump from one read to another while non existent in the real sequencing data.

details about the advantages and drawbacks, as well as worst case complexities for these algorithms, can be found in the lectures [182].

These drawbacks can all be addressed as we will see in section 4.5.4. The main advantages, fast creation of the graph, and bounded memory usage (bounded by genome length and not linear to coverage), as well as the regular structure of the graph (each node is a sequence of fixed length, each edge is an overlap of fixed length) were the reasons why it was chosen as a base for the custom assembly algorithms used in this work.

4.5.3 Scaffolding

Both OLC and DBG assembly methods generate contigs, these contigs can be further threaded together using the information provided by linked reads e.g., short read pairs or long-range (10x Genomics) information, as illustrated in figure 4.14. If read pairs

map to two contigs the approximate distance between the contigs can be inferred from the fragment lengths the reads came from. The fragment length is dependent on the template and sequencing technology and the expected distribution of fragment length is known and therefore the maximum and minimum possible (probable) lengths are known.

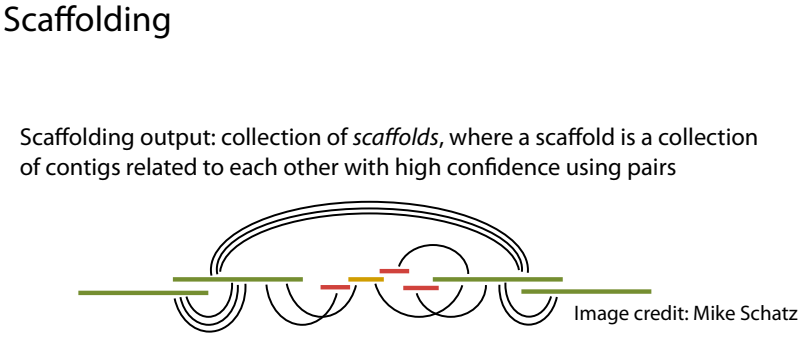


Figure 4.14 Scaffolding illustration *Slide by Ben Langmead [182]*

This extra step allows to put contigs together to create a bigger assembly even if sequence between the contigs may not be known, or at least not resolved at a per base resolution.

4.5.4 Custom De Bruijn Graph based Assembly

The assembly algorithm used to solve variants in this project is based on the De Bruijn Graph approach. For a given region that is overlapping the variant of interest (that is to be solved) all sequencing reads are extracted as are their mates (which may be unmapped). These sequencing reads are then transformed into k-mers to build the custom DBG. The difference between the custom DBG and a traditional DBG is that the custom version is extended with look-up tables for the nodes and edges, these look-up tables are implemented as hashtables and have an entry for each edge and node. The values for each entries are meta-data related to the edge or node. This meta-data allows for new pruning, traversal, statistics algorithms to be run on the graph. For example a possible meta-data for the nodes (k-mers) can be a histogram of positions where the k-mer has been mapped to the reference, this information is extracted from the read at the same time the k-mer is extracted and added to the table. This way it is for example possible to infer the most likely position or set of positions of any given k-mer. These tables can be seen as attributes of the edges and nodes. However, they are stored in separate data structures in order to make the traditional algorithms like graph traversals and topology analysis as efficient as with the traditional de Bruijn graphs. This makes the generated graphs also completely compatible with the ordinary approaches. These extra tables make many new algorithms and approaches possible in a totally transparent way.

Assembly algorithms are often reference-less and are used to generate contigs from raw sequencing reads which do not provide much more information than the sequence itself. This is the reason why traditional DBG approaches do not have any other extra meta-data. However, when working with mapped reads as is the case in this project,

there is extra information available besides the sequence. For example, the mapping status, mapping quality, position(s) on the reference, local coverage levels, reference to mate read, attributes of the mate such as its position, inferred fragment length, edit distance to reference, to name a few.

These meta-data allow for example to directly map unambiguous k-mers back to the reference, which may be used to split the graph in subgraphs based on these anchor points. Another useful application is to trace back the reads that generated the given k-mers that are removed during the pruning phase of the algorithm.

The creation of the graph is basically the same as the traditional DBG approach except for the two added steps ($*$) in algorithm 4. The output of this algorithm can be interpreted back to a standard DBG by ignoring the two extra meta-data property tables.

Algorithm 4 Custom De Bruijn Graph Creation Algorithm

Require: Ω : Genomic region that spans the variation event
  DBG : $<nodes, edges, node_{properties}, edge_{propertiess}> \leftarrow <\emptyset, \emptyset, \emptyset, \emptyset>$
  for all r such that $(alignment(r) \cup alignment(mate(r))) \cap \Omega \neq \emptyset$ **do**
    for all $(k+1)_{mer} : k_p$ in r **do**
      $k_1 \leftarrow k_p$ without last base
      $k_2 \leftarrow k_p$ without first base
      update $nodes$ of DBG with $\{k_1, k_2\}$
      update $edges$ of DBG with $(k_1 \rightarrow k_2)$
      update $node_{properties}$ of DBG with $\{properties(k_1), properties(k_2)\}$ $(*)$
      update $edge_{properties}$ of DBG with $properties((k_1 \rightarrow k_2))$ $(*)$
    end for
  end for
  return DBG

For this project the data structure used for the graph and meta data was as follows, the graph was represented as an adjacency list, as is done in the boost library[3]. This is an efficient representation for directed graphs and using the same representation as the boost library allows to apply the boost algorithms directly to our data structure if needed.

The meta-data tables are hash tables that link nodes, resp. edges, to their properties, which is a collection of an abstract type of property. This allows to redefine the abstract type to any data type and the reason a collection is used is because a node or edge can have multiple properties.

Access to the properties for any given node or edge is therefore of constant complexity relative to the size of the graph. The property can be as simple as a reference to the read of origin in the input assembly file, all other properties can then be extracted from this reference later if necessary. The exact data that is stored can be changed later in order to optimize the performance of the algorithms.

[3] `https://www.boost.org/doc/libs/1_72_0/libs/graph/doc/adjacency_list.html`

Extra possible algorithms based on this representation

Filtering and pruning the graph based on a property We now have possibilities to trim the graph based on the weight of its edges and its topology. With the basic approach edges that have a low weight are removed (weight based) as are islands (topology based). With the extra properties nodes and edges can be filtered based on any predicate, such as mapping quality below a threshold. This could of course also be achieved by pre-filtering the k-mers (nodes) before or during the graph construction, Anyway now that the graph is already constructed it is possible to generate "views" (pruned or altered version of the graph without altering the graph of origin) of this graph based on these properties and combinations of these properties, which would have been more difficult during graph construction. E.g., remove nodes of good mapping quality that have neighbors of bad mapping quality above a threshold.

Graph traversals based on properties All the possible graph traversal (walks) applicable to the DBG (a directed graph) can now use the extra properties to create novel traversal algorithms. E.g., block paths (ignore edges) based on properties.

Anchor point extraction For every node it is possible to extract all the positions of origin of the k-mer and their frequency. From this it is possible to create an histogram of positions for each k-mer and extract the most likely positions or unique positions as anchor points for the nodes. Unique positions of origin map nodes (k-mers) directly to a position on the reference, this is helpful to rapidly map the contigs generated by the graph back to the reference or study more complex variations such as rearrangements.

Extracting genotypes

Genotypes (contig sequences) are extracted from the De Bruijn Graph by extracting all the non looping paths from starting nodes (no incoming edges). If the graph has a loop the loop will be taken once and the path will end. This allows to generate smaller non repetitive contigs for the graph. These contigs can be realigned to the reference and allow to solve some parts of the region. If the graph has loops it means there is some kind of repetition in the sequence, this cannot be directly solved by the graph itself and therefore will results in sequences that only span the non repetitive regions (relative to the k-mer size chosen). The number of repetitions can be estimated based on the coverage. For small repetitions or small novel insertions they can sometimes be solved with the extra properties of the graph. For example by using the read of origin information and traversing the graph with paths that are coherent with the reads or pairs of reads.

Mapping back to the reference

Since the assembly was done with several reads that came from well mapped and aligned regions around the event they can be used as reference anchor points for the generated contigs. Therefore mapping back to the reference can be done easily by taking the ends of the generated contigs. If these ends contain enough k-mers (nodes) that are uniquely mapped the contigs can be directly mapped back to the reference at that position. The generated contig is therefore well anchored to one or both sides of the event and this is often enough the correctly map the contig to back to the reference.

In complex cases or interchromosomal translocations the contig may be anchored on two different regions on each end. This is not a problem but means it may need to be mapped and aligned to two places for analysis.

Once the mapping has been done it is important to realign the contig to the reference in order to solve the variant in more details.

Realignment to reference

Realignment to the reference is done with the Needleman-Wunsch algorithm [183] and a specific scoring scheme. The scoring scheme (scoring matrix) requires to be adapted for the type of variants, especially when considering insertion or deletions scores and gap continuation penalties.

Realignment is only done when both ends are well anchored (mapped) to the reference over a reasonable size span. For example, it is not recommended to realign a 1000 base pair contig over a 20'000 base span on the reference because of a 19'000 base deletion. There are better heuristics to solve such cases, e.g., simply by comparing the length of the contig and distance between anchor points (mapping of both ends on the reference).By doing this, insertions or deletions can sometimes be directly be inferred based on the relation between the length and distance between both ends. E.g., if both ends span a region bigger than the assembled contig there probably is a deletion and if both ends span a region smaller than the length of the contig there probably is an insertion. Particular care has to be taken in case of graphs with loops. Cases where graphs have loops can be marked and left for further analysis. Graphs that have no loops (they may have bubbles) can generate contigs that can sometimes directly solve the region. Bubbles are events where two possible paths are possible in the graph. Most of the time they appear because of differences between the maternal and paternal alleles.

4.5.5 Summary on assembly

Assembly is a complex subject and requires specific care in order to give good results. The strategy taken here is to use a de Bruijn Graph based approach that links the k-mers with the reads of origin through a table, table that is optional and omitting it leaves us with the traditional graph. Having this extra table allows to extract all information related to the reads if necessary and allows to counter some drawbacks of the de Bruijn Graph such as the apparition of paths that are not existing in the reads (because the same k-mer appears at multiple locations), graph traversals can now be done coherently with the sequencing reads if needed. The drawback of this extra table is that it grows linearly with the number of reads used to create the graph. The graph itself still grows sublinearly relative to the number of reads. In most cases this is not really a problem because for SV detection only local assembly is done and this is for "small" regions (not the entire genome or chromosome). Chapter 7 will show some examples of assembly graphs and generated contigs based on the augmented de Bruijn Graphs.

5 | Design

Contents

5.1 Introduction

The design came out of an iterative process and was improved in order to provide a flexible and modular framework, as to augment the individual tools value and usability. The design of the framework will be presented in this section.

The goal of this project is to create tools for CNV assessment of whole-genome next generation sequencing data with the main idea to go from an input alignment file to output a list of breakpoints defined regions of interest and to define regions with CNVs.

Performance was also a major concern during the design of this framework, the idea was to be able to process whole genome sequencing data at average coverages of 30-100x in a reasonable amount of time on consumer hardware. In order to satisfy this the system was designed with a general mindset of balancing the cost of the algorithms used and the sizes of the input data sets as roughly shown in figure 5.1.

This means the algorithms that will be applied genome-wide will be very simple and more complex algorithms such as local assembly are only applied to a small subset of specific cases.

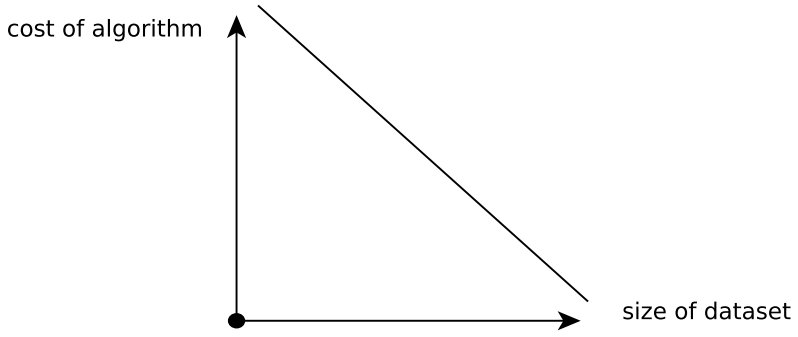


Figure 5.1 Balance between cost of the algorithms and size of data set

5.2 Architecture

The architecture was chosen to balance the cost of the algorithms with relation to the size of the data sets. First, simple algorithms are applied on big data sets, then followed by filtering and prediction algorithms that will reduce the data set. Finally, a collection of specific algorithms, that can be much more involved, are applied on the remaining cases.

In order to understand the architecture of this project it is important to study the general anatomy of variant callers.

5.2.1 Anatomy of a variant caller

A variant caller can be seen as an entity that takes an alignment file as input and outputs variant calls. This is illustrated in figure 5.2. A variant caller can be tailored to output only a specific type of variant, e.g., SNPs or deletions or a more broad type of variants such as CNVs or SVs.

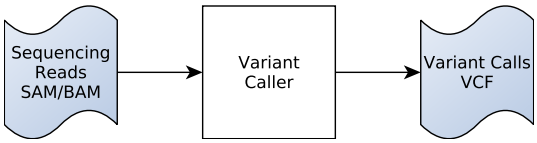


Figure 5.2 Variant Caller overview

The idea of the design approach that was taken in this project was to decompose the variant callers into subparts, to analyze their anatomy. Figure 5.3 shows an hypothetic generic anatomy of variant callers.

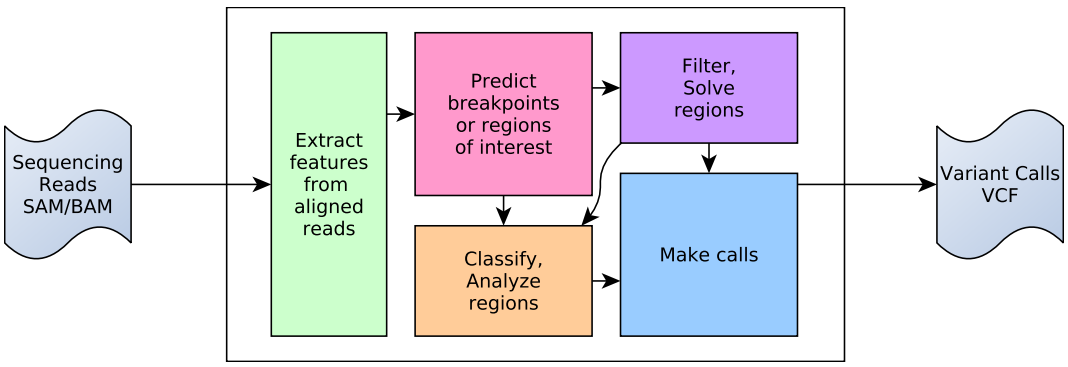


Figure 5.3 Variant Caller anatomy

Different variant callers could be split to fit this generic anatomy as shown in figure 5.4 were different hypothetical callers have the same basic anatomy but the subparts differ in their specific function but have the same overall role.

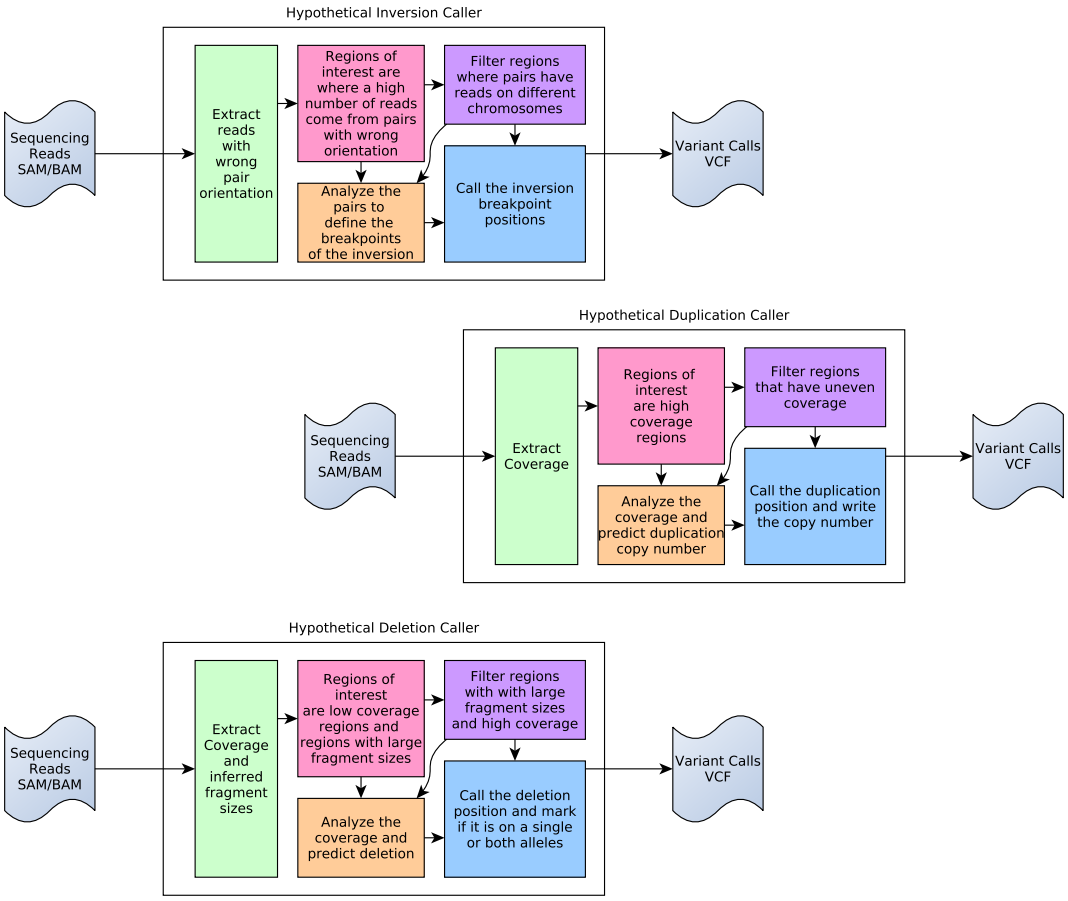


Figure 5.4 Hypothetical variant callers

Even if their internal workings may be somewhat different, callers share several internal pieces, for example all callers will extract features from aligned reads, and as was discussed in the analysis chapter as well as in the algorithms and strategies chapter, several calling algorithms use the same features. These features include coverage, fragment length, clipped reads etc. Composite callers that will call more generic variants such as CNVs or all SVs will share internal parts with their more specific cousins. Therefore by thinking about the anatomy of callers and by dissecting them, it is possible to create generic building blocks to create callers. By doing so, we create basic building blocks that can then easily be reused or adapted to create any caller. The basic blocks of this hypothetical anatomy are discussed below.

Signal extractor

The first stage (green box) extracts signals of interest out of the alignment file. These can be first order signals directly extracted from the assembly such as the coverage or second order signals (i.e., processed signals) such as edge detection in the coverage signal, filtered or combined versions of the first order signals. Therefore, the signal feature extraction block it is composed of a signal generator block and possibly a signal

processing block. The signal generator block relies on the algorithm presented in section 4.2.1 and the specific increment and decrement function used will define the signal generated.

Since so many callers rely on the same signals it was chosen to create a tool that can do signal extraction on its own. This tool will extract the feature signals and the callers will load the signals instead of generating them themselves. This has the benefit of only needing to generate, compute, the signal once. Once extracted or generated, then any subsequent caller can use it. This way, if several callers rely on the same signals they can use the already existing signal instead of wasting time to redo the same extraction inside each caller. Another reason the signal extract was made into an independent tool is because the extracted signals can be of interest for other downstream analyses or as input to other programs. Also once they are generated for a sample, the subsequent stages can be run as many times as required without having to wait for the signal extraction again. The signals can also be of use for analysis of the alignment file by hand and when displayed in IGV.

Predictors

The next common stage is the prediction of breakpoints and regions of interest, this stage is very important because it tells the callers where to look. Of course, it could be possible to look at all the data but this would be a waste of time and not suited for whole-genome analysis.

Predictors simply predict breakpoints or regions of interest based on the extracted signals and features of the assembly. They will guide the subsequent analysis tools to regions that seem promising.

Breakpoint predictor A breakpoint predictor takes as input one or more signals and outputs a sparse signal, i.e., a list of positions, where it predicts a breakpoint. This signal can have an extra polarity attribute, discerning between a starting and ending breakpoint of an event or be a plain positional signal.

An example of a breakpoint predictor could take a coverage edge detection signal (list of high increases or decreases in coverage) and output starting breakpoints when the coverage increase is above a threshold and ending breakpoints when the coverage decrease is above a threshold.

The interpretation of the polarity would be dependent on the event, a drop in coverage could be a starting breakpoint for a deletion event, but could also be an ending event for a duplication event. Therefore the extra polarity attribute should not necessarily be interpreted as starting or ending but can be interpreted this way. In any case, if an edge detection signal is used, it is often helpful to analyze the polarity of the edge (increase, decrease).

Another example would be to predict breakpoints based on the number of reads that have an unmapped mate or mate mapped on another chromosome.

Region predictor A region predictor takes as input one or more signals and outputs a sparse signal, in the form of intervals instead of positions.

An example would be taking the coverage signal as input and outputting the regions, intervals, were the coverage is under or above a threshold.

Filtering and solving of regions

The filtering and solving block is much more specific it serves as a pre-processing stage to the next block in the way that it will filter out specific regions that were predicted by the previous stage because they are not relevant or do not satisfy a property. However, in some cases it can also be used to solve some regions directly without further analysis. This boundary between this block and the next is a bit fuzzy. Therefore, the ability for this block to also directly make calls without further analysis while filtering was left as an option, for cases where enough information is available during the filtering process to make a call without the need for further analysis.

Classify and analysis of regions

This block now has a (filtered) collection of regions to analyze. The number of regions in this collection is usually very sparse on the genome and this stage will usually query reads from the assembly to do an in-depth analysis. This block is very specific to the type of event being called. Since the number of events to be handled by this block is usually low, more complex analysis algorithms can be run here.

Making calls

The final common block is there to generate the output calling file, it will generate the variant calls based on the results of the previous stages and create a variant call entry with all the required information to be written to the output file.

5.2.2 Design and decomposition

In order to create a generic solution to build callers their individual parts have been broken down and decomposed into function. This resulted in separate tools and a framework of functions and algorithms for caller *reconstruction*. Figure 5.5 shows the design and decomposition chosen for this project. On top there is the generic signal generation and processing tool (feature extraction). This tool allows to extract signals from the assembly file and reduces the need for the individual callers to go through the entire alignment file to generate the signals themselves. This tool can also work as a stand-alone tool to generate signals from an assembly file for other analysis purposes. Below the signal related tools is the calling and prediction framework, it is comprised of a collection of generic algorithms and functions that can be specifically tailored and recombined to create variant callers. Such a (re)constructed caller will then use the signals it requires as well as possibly the input alignment file to output the variant calls.

The callers that will be created with this project will have a very similar architecture to existing callers with the exception that the signal feature extraction has been made into a separate tool. The framework architecture is a collection of tools based on their purpose and will serve as building blocks for the specific entities.

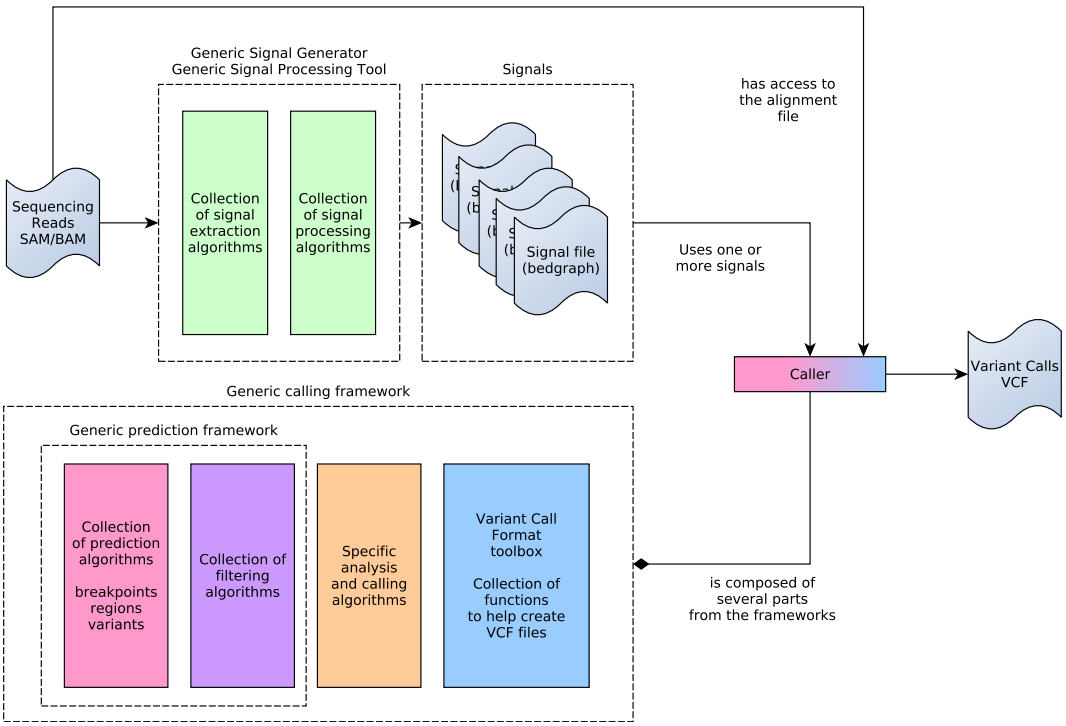


Figure 5.5 *Variation calling framework and tools*

5.2.3 Arbitrary architecture by recombination

This decomposition makes it possible to create other forms of pipelines, analysis and prediction tools of other shapes and architectures. This also allows to to export signals and predictions or call to other tools or, the other way around, import calls from other tools to use in our framework. Figure 5.6 shows a hypothetical rearrangement of tools and algorithms from this project to create a new pipeline.

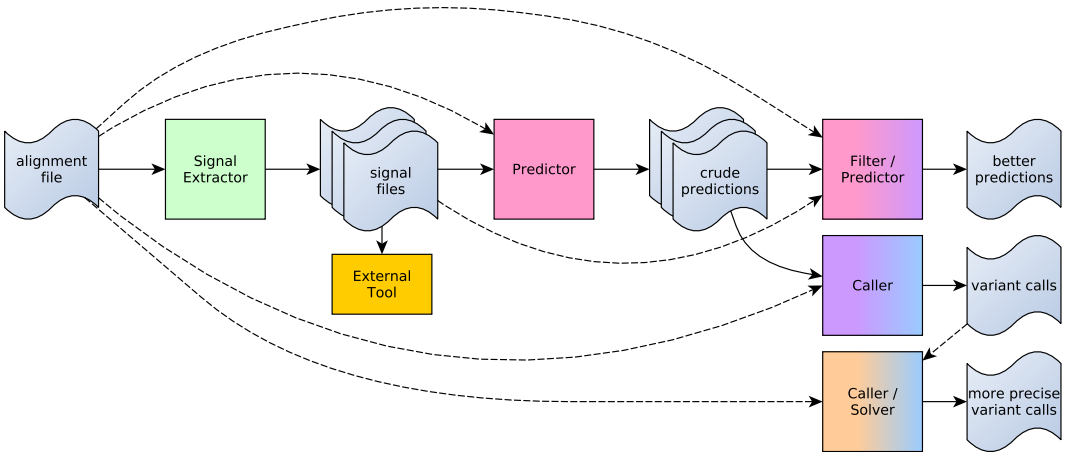


Figure 5.6 *Possible rearrangement and new pipeline created from the collection of algorithms in the frameworks and tools designed for this project*

This decomposition is flexible enough to both create new tools and accommodate any new type of input file or combination of existing processing stages.

5.2.4 Visualization and benchmarking tools

In order to develop and debug the different algorithms and tools for this project it was important to be able to visualize results both of internal stages as well as output files. All the input and output files could be visualized with IGV however it requires to manually browse the files. Therefore as part of the design of this software framework a collection of tools to visualize data or benchmark results was also designed.

The idea is to be able to load results in IGV and control the viewing in an automated way from any point within the software.

For other type of data or intermediary data tools were made to display it. For example assembly graphs for the local re-assembly algorithms (which are part of variant resolution and analysis). This also includes tools to generate benchmarking results as tables and statistics, e.g, for variant caller benchmarking.

Incorporating the visualization and benchmarking tools directly into the software framework allows not only to call these tools at any point where needed, but also allows to make the process repeatable and automated. These tools will help development, debug, and benchmarking of the whole software.

Visualization in IGV

Figure 5.7 shows the idea to provide a function to display any data, any input file, in IGV and have an interactive session from within the software.

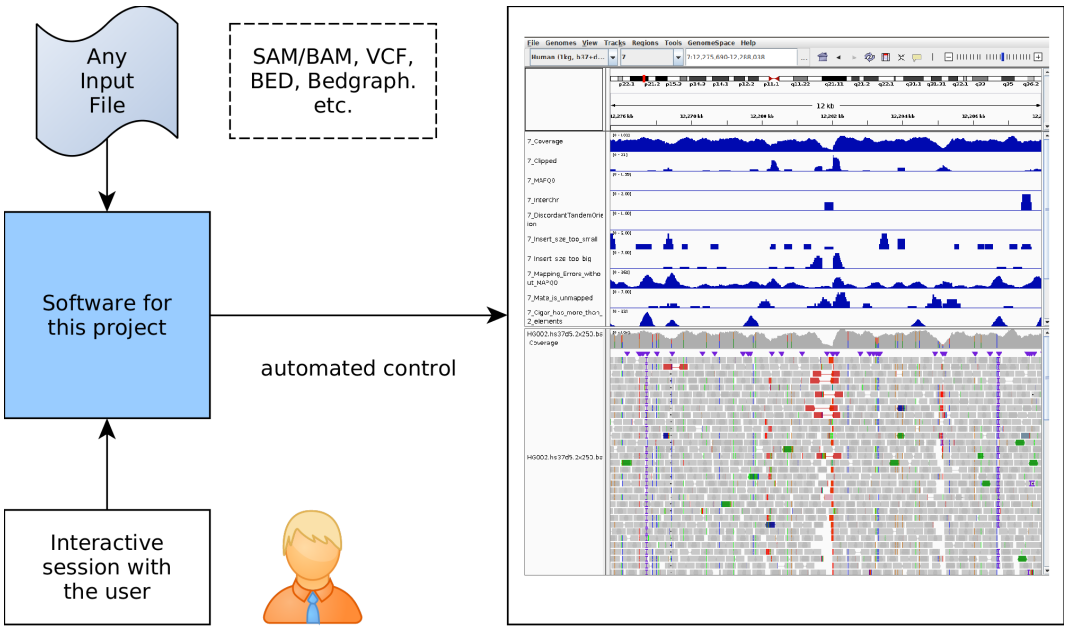


Figure 5.7 Interactive session with IGV for debug, development, and visualization

This software component will take as input genomic regions (locations on the genome) and will automatically control IGV for display purposes. It will also provide a basic interactive session for the user. This functionality should be able to take any input data collection, and with a provided function that convert samples from that collection to a genomic location, control IGV. Therefore it can be used at any time with any type of data given the user provides a conversion function for the data type to a genomic location.

Visualizing the features, signals, alignments files, from any point in the software makes it much easier to understand how particular cases are handled by the algorithms and allows to think about the analysis.

Visualization of assemblies

The local de-novo assembly tool is a complex part of this project and may require debugging sessions as well as exploration of novel algorithms on the augmented de Bruijn Graph. Therefore the system is designed to be able to output assembly graph data directly to visualization software.

Figure 5.8 shows the idea of displaying the assembly graph from any stage of the assembly algorithm inside the software, this can be optionally interactive. Having this possibility will help experimentation and development of assembly algorithms tremendously.

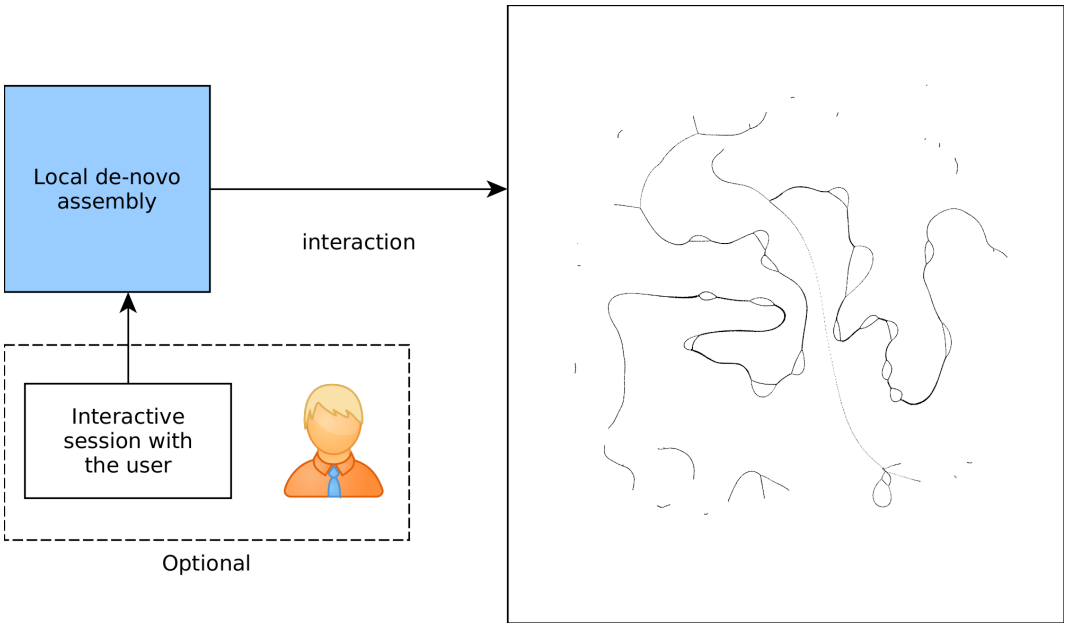


Figure 5.8 Visualization of assembly graphs

Combination of this tool and the previous visualization tool that controls IGV makes it possible to both see the assembly graphs and see the aligned contigs to the reference genome making this software collection of tools a very potent development method for new assembly related algorithms.

Integrated Benchmarking Tools

Benchmarks functions for this project will be included in the same software framework in order to be able to run them when needed and in an automated way.

Benchmark tools will include the measure of time of any function call of the software, this will give a simple but effective way of measuring the time it take to call any part of this software or any tool developed with the framework.

Benchmark tools will also include automated comparison with other variant call files and truth sets and include metrics that define if two variant calls are close enough to be considered the same call. These tools will also generate the data for the results chapter of this thesis. Figure 5.9 shows the design for the variant call benchmark tool.

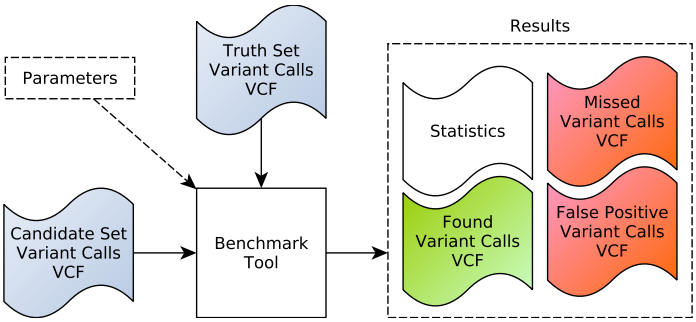


Figure 5.9 Variant Call Benchmark tool overview

The tool takes as input a truth set variant call file and a candidate variant call file. The tool will then evaluate the candidate variant call set against the truth set. The tool will check every entry in the truth set and look if there is a candidate call that represents the same variant. The tool will output the statistics in terms of true positives (candidate calls that are in the truth set), false positives (candidate calls that are not in the truth set) and false negatives (truth calls that are not in the candidate call set).

Not only will the statistics be computed but three extra variant call files are generated. The found variants (true positives), the missed calls (false negatives), and the wrong calls (false positives). These files can then be studied to understand why a particular caller performed well or poorly.

The benchmark tool comes with an input set of parameters which allow to define how to extract the interval of a variant call, usually the breakpoints around the call. This is also so that different calling formats can be normalized, because some use start + length, others use start + end positions etc. The parameters also include a function that tells if two intervals of variant calls are considered close enough to be considered the same call. This allows to pass different type of comparison functions, usually reciprocal overlap is used. By leaving this generic not only makes it possible to set the minimal reciprocal overlap threshold (e.g., 80%) but also allows to use other functions, e.g., ones that compare predicted insertion sequences in the case of insertions or other properties such as copy number for duplications.

This tool will allow benchmarking two variant call sets with any metrics that are relevant to the calls and with different levels of rigorousness when comparing calls. The extra generated output VCF files are very helpful to use with the visualization tools above to understand why calls were missed or made. This will provide a feedback loop that will allow the developer to improve the variant calling tools as depicted in figure 5.10. Combining a benchmarking tool with the visualization tools will make it much easier for the researcher to understand the results of the algorithms and get a deeper insight of the impact of the different signals and features extract from the sequencing data. The statistics could also be used to create a feedback loop for a machine learning predictor based variant caller as depicted in figure 5.11

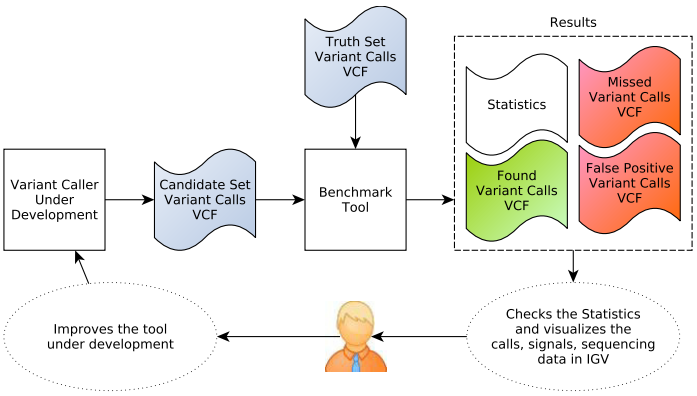


Figure 5.10 Variant Call Benchmark tool in the development feedback loop

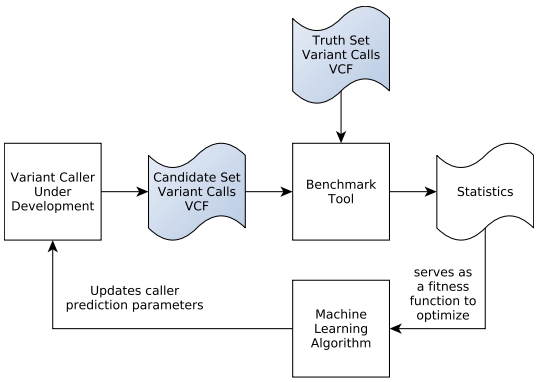


Figure 5.11 Variant Call Benchmark tool for machine learning based predictors

This benchmark tool will serve as a basis for quantification of result quality of variant callers and as a handful tool to facilitate the development of callers. It is designed to be flexible in the type of data (statistics, files) it outputs so that it can be used in as many situations as possible. A possible scenario will be to automatically generate the result tables used for this report.

5.3 Summary of Design and Architecture

The very generic and modular architecture and design choices made for this project are based on the need to be able to experiment and test as many approaches as possible. The framework and tools developed during this project were designed to make exploration and experimentation as easy as possible. Giving not only the possibility to benchmark all the explored solutions, but also be helpful to optimize them to any need. The next chapter will go over the implementation choices for the different parts introduced in this chapter and chapter 7 (tools) will show more specific tools that arose from this project and arbitrarily free form of architecture and design.

6 Implementation

Contents

6.1 Introduction

This project was implemented in the Scala programming language. Scala combines object-oriented and functional programming in one concise, high-level language. Scala's static types help avoid bugs in complex applications, and its JVM and JavaScript runtimes let you build high-performance systems with easy access to huge ecosystems of libraries [184].

`https://www.scala-lang.org/`

The possibility to execute Scala code in a interactive environment is also a very strong development and debugging tool. The JVM runtime also provides the possibility to access all of the Java libraries making it a perfect tool for bioinformatics. Since Scala runs in the JVM it is possible to port to any system, and is also able to scale in distributed computing platforms, e.g., with the Apache Spark lightning-fast unified analytics engine [185]. It is also possible to call native libraries (written in C or C++ for example) through the Java Native Interface (JNI) as well as to call other software such as Samtools, GATK, Picard, written in any language to execute a specific task.

The choice to use Scala as the programming language was also motivated by the fact that the Genome Center chose Scala to build their data analysis and interpretation pipeline.

The version of Scala used to develop the software for this project is : 2.13.0

6.2 Libraries

This section introduces the external libraries used within the project.

6.2.1 HTSJDK

HTSJDK is a Java API for high-throughput sequencing data (HTS) formats. It was developed at the Broad Institute and is used extensively in GATK and IGV proving it's maturity and usefulness.

The library is open-source and most of its source code is under MIT license. It offers a unified Java library to access common file formats, such as SAM and VCF, there are also a number of utilities for manipulating HTS data. The source code is available at :

`https://github.com/samtools/htsjdk`

Integration in Java/Scala projects can be done through the Maven repository available at :

`https://mvnrepository.com/artifact/com.github.samtools/htsjdk`

This library (version 2.21.1) was used to read files in SAM/BAM format and access or extract all the data related to the alignment records. It was also used to generate output alignments in SAM/BAM format. It was also used to handle reference FASTA files.

6.2.2 BioJava

BioJava is an open-source project licensed under LGPL 2.1 that provides a Java library for processing biological data [186]. The project aims to simplify bioinformatic analyses by implementing parsers, data structures, and algorithms for common tasks in genomics.

The library website is :

`https://biojava.org/`

The source code is available at :

`https://github.com/biojava/biojava`

Integration in Java/Scala projects can be done through the Maven repository available at :

`https://mvnrepository.com/artifact/org.biojava/biojava-core`

This library was solely used for the Needleman-Wunsch global alignment algorithm and scoring matrix implementations.

6.2.3 Apache Commons

Apache Commons is an Apache project focused on all aspects of reusable Java components. Commons is dedicated to one principal goal: creating and maintaining reusable Java components. This is a place for collaboration and sharing, where developers from throughout the Apache community can work together on projects to be shared by the Apache projects and Apache users.

`https://commons.apache.org/`.

It includes several packages for math, text, and system utilities.

The Statistics package was used to handle statistic computations such as extraction of the read length distribution percentiles. This is the same package that is used in IGV for statistic computations. It is released under the Apache 2.0 license as are the other packages in the Apache Commons project.

6.3 Software implementation of this project

The project is structured into packages, which are a collection of tools and functions. These packages are implemented in the form of Scala objects for the scope of this application but could be implemented as a package (in the Scala/Java sense) together if to be used by another software or directly used as-is by including the source code.

6.3.1 Software packages developed during this project

This section gives an overview of the software packages developed for this project. They are contained in Scala objects and provide functions to do or implement specific tasks.

General packages

- **Basic**

  The basic package provides basic interaction with files and general input output. It provides functions to generate a stream that allows to read files line by line from either compressed (gzip) or uncompressed files. It also provides an output stream to write output files either uncompressed or compressed. This makes the handling of input and output files transparent to the developer, the correct processing will be applied based on the file type chosen, e.g., if the user provides a path to a compressed input file, the decompression will be applied to the stream.

  This package also provides a function that allows to count the time used by any portion of code, this should be moved to a benchmarking package but was implemented here because it was a very basic and useful function used throughout developments to profile the time required by the execution of any code block.

- **BedSignalToolBox**

  This provides functions to read (extract information) and write BED and bedgraph format files. It also provides basic algorithms on the BED data such as filtering or extracting data based on given properties, as well as, conversion algorithms to convert BED encoded data to other formats and the other way around.

- **CommonGenomics**

  This package provides data structures to hold commonly used genomic data (e.g., loci or regions) and basic algorithms to deal with these data, mostly genomic region related, intersecting, merging, sorting algorithms.

- **RegionSamToolBox - SamToolBox**

  These two packages provide functions to handle SAM/BAM input files and extract records based on reference location. They also provide filtering capabilities to handle large collections of records.

  The packages also provides useful extraction functions, such as extracting the names of the regions of the alignment file, extracting statistics such as the distribution of inferred fragment sizes, extracting the average read length and its uniformity. The packages also provide functions to create SAM/BAM record files, for example in order to write alignments of contigs generated by local re-assembly.

  The two packages rely heavily on the htsjdk library that provides the low-level functions to parse, read, and write files in the SAM/BAM format.

- **VcfToolBox**

  The VCF toolbox is collection of utilities to handle VCF files and variants, both reading and writing VCF files as well as providing a data structures to hold VCF entries, generate them, filter and convert them.

  This package also provides generic transformation algorithms on VCF files such as filtering them given a predicate, this allows for example to quickly generate a new VCF file with only the entries with the "SVTYPE=DEL" parameter or filter VCF files so that only the variants with the "PASS" entry remain. Other functions include re-identifying variants (giving them a new ID) or sorting them by any property (e.g, size).

- **EndToEnd**

  Implements some end-to-end functionality such as calling variants from input files and generating the final output variant call file.

Prediction and Calling packages

- **Predictors**

  This package includes functions to help create predictors as well as predictors to predict breakpoints, regions, and variants given the input signals.

- **ImprovedPredictors**

  The improved predictor package holds more advanced predictors such as insertion predictors, copy number predictors and interchromosomal breakpoint predictors. This package also includes variant calling algorithms. These include deletion, insertion, inversion, and duplication callers.

Assembly packages

- **Alignment**

  The alignment package provides functions to align sequences such as the Needleman-Wunsch algorithm, helper functions to generate the CIGAR string for alignments, as well as, functions for generating SAM records of aligned sequences. It also provides functions to directly re-assemble and re-align mapped reads from regions of the BAM file and produce alignments for the assembled contigs.

- **Assembly - AssemblyGraph**

  These two packages provides the functions to create the de Bruijn Graphs from sequencing reads as well as update and prune assembly graphs, they also provide more advanced functions such as generating contigs, paths, possible mapping positions and other properties from the assembly graphs.

- **Graph - GraphG**

  This package (GraphG, Graph is the non generic version) contains abstract graph related data structures and algorithms. This package implements graphs in a generic manner. It implements graphs as generic adjacency lists and is based on the same ideas behind the C++ Boost graph library. As with the C++ version with templates this implementation allows for the nodes, edges and other properties to be of any desired type. Every algorithm is generic and will work with any data type. This package allows for example to create graphs were the nodes are integers, k-mers, SAM alignments or any other type. This not only allows to create any graph but also allows for performance optimizations such as creating a graph with a very simple data type such as an integer that serves as an unique identifier and having a table that links that unique identifier to any other property. This allows to apply all graph related algorithms that do not require the extra properties on a very simple graph and achieve good performance results. The choice is left to the user based on the specific needs.

  The augmented de Bruijn graphs for assembly were created this way with graphs that had nodes either being k-mers represented by unique identifiers and augmented with tables (hashtables) of node and edge properties. This allowed to run all the graph algorithms on a simple integer graph and still be able to run the more advanced algorithms such as generating contigs, extracting histograms of possible positions of provenance for the k-mers, querying the reads of origin using the tables and this in an efficient manner.

  General graph topology algorithms are also provided here, e.g., find nodes that have only incoming or outgoing edges, find non-looping paths, extract the longest paths from a graph to name a few.

  More abstract algorithms are also given such as pruning nodes based on a predicate, which will apply a predicate function to all nodes, remove them if they fulfill it and finally remove unconnected edges.

  This package was designed to be compatible in data structure with the C++ Boost graph library and this allows to call the algorithms implemented in that library through the Java Native Interface (JNI). This gives direct access to validated and very optimized implementations of graph algorithms such as Dijkstra's Shortest Paths, Connected Components, Kruskal's Minimum Spanning Tree algorithms, and others, if required[1].

Visualization packages

- **GraphToGephi**

  This package allows to send graphs created with the graph package introduced above to the program Gephi [187] over the network. This interaction makes it possible to dynamically visualize graphs and display their properties as well as

[1] `https://www.boost.org/doc/libs/1_72_0/libs/graph/doc/index.html`

change their layout. Gephi was chosen because it is open source and very effective at displaying graphs of a very high number of nodes and edges. The assembly package also provided some basic functions to export graphs as .dot files which could then be printed with graphviz [188], however graphviz does not handle big graphs or dense graphs very well.

- **CheckingVariantsInIGV**

  This package provides the functionality of reading VCF files or any collection that can be converted to a genomic region and interacting with the user in order to send commands to IGV to display the variants or regions. Functionality includes zoom factor around the event, interactive search for events in a specific region (chromosome), being able to skip events etc.

- **Things**

  This package ought to be renamed but for now handles everything related with the network connection to Gephi and IGV, it provide functions to connect to these programs at the socket level and send data through these sockets.

- **Graphml**

  Contains a single function to generate a graphml format representation of a DNA sequence from a string in order to generate diagrams with tools such as Dia or Yed. This function was used to help create the figures in chapter 2.

Benchmarking packages

- **Basic**

  The basic package, introduced above, did provide the time related profiling function that can evaluate the time required by any code block execution.

- **BenchMarkGraph**

  This is a benchmarking package that provides tools to evaluate results from this project. It can check reciprocal overlap between variant calls, evaluate the statistics of relationships between variant call files. These functions, given a truth set of variant calls, can extract all the statistics in terms of true/false-positives/negatives. It also provides functions to analyze the distances between calls that may be related in order to quantify the differences between callers.

  This package also provides functions to generate data for graphs and tables.

6.3.2 Main program

The main program resides in the "Canevas" package (Canevas.scala source file). This is a placeholder name that was chosen because canevas has the meaning of "rough sketch" or "outline" in French but also is a play on the acronym CNV and is very similar to the name chosen by the CNV caller "Canvas" by Illumina.

The functions of the main program are :

- To implement the signal generator to generate signals from the input whole-genome alignment BAM file.
- To do predictions given the signals (breakpoints, regions of interest).
- To call variants (run callers created using the framework).

Since this is meant as a software framework separate "programs" can be created from this collection of tools, functions, and algorithms. Different tools that derived from this framework are demonstrated in chapter 7. These tools include, signal generators/extractors, predictors and callers, a local de-novo assembly tool, and a variant photography tool.

6.4 Implementation choices

This section goes over some of the important implementation choices made during this project.

6.4.1 Reproducibility and Portability

With scientific computations, reproducibility is important, the computations should rely as little as possible on random parameters. In this project this is not a problem since we only extract information from a fixed input file. However, another source of problems with reproducibility is library dependency and system specific parameters. In order to be able to run the program under known conditions it is packaged into a Docker image making it possible to run on any system that supports Docker.

As for imported libraries the version of the imported library is fixed so that unless the developer willingly changes this, the program will always be run with the same version of the libraries. All the external libraries are imported in the Docker image with a fixed version number. This removes the burden from the user to install libraries and dependencies on which the program relies as well as makes sure the same coherent system is used every time.

6.4.2 Compression

Whole Genome Sequencing data is big, often in the hundreds of gigabytes, therefore compression is key. Having to decompress compressed files for processing can be a major pain and may take up unnecessary disk space. Therefore this project can take

"gzipped" (GZIP compressed) [189] files as well as uncompressed files as input. This is implemented in a transparent manner for the user and all output files can be generated GZIP compressed as well. This is implemented as a compression stream, therefore the whole decompressed file never needs to be entirely loaded in memory or on disk.

For example a per base read count in BED format for human chromosome 8 takes 1.7 GB while the compressed file takes less than 400 MB.

The handling of compressed files requires a small amount of extra processing power, however this is counterbalanced by the fact that the compressed files are lighter and therefore require less data transfers from disk which is often one of the major bottlenecks, especially with non solid-state (mechanical) drives.

Building the program as an entity that directly reads and outputs compressed files not only reduces the disk usage footprint but can also improve performance because of this. Therefore, by default it handles all input and output files with compression, the program is however compatible with uncompressed input files and the user can always decompress the output files at his leisure.

6.4.3 Parallelism

Parallelism was implemented for the most time consuming tasks inside the project. Naturally, the CNV assessment can be run for different samples in parallel by executing multiple instances of the program. Inside the program, the tasks are separated by chromosome (more generally by reference contig). The chromosomes are sorted by size and the tasks for the biggest chromosomes are launched first to optimize the overall runtime. Several sub-tasks can be run in parallel and some tasks have further parallel implementations. This is mostly used for testing tasks rapidly during development, during normal execution, the tasks are only parallelized by chromosomes. The level of parallelism can be set by changing the parameters of the task.

Parallelism was implemented using *Futures*. From the Scala documentation, "Futures provide a way to reason about performing many operations in parallel – in an efficient and non-blocking way. A Future is a placeholder object for a value that may not yet exist. Generally, the value of the Future is supplied concurrently and can subsequently be used. Composing concurrent tasks in this way tends to result in faster, asynchronous, non-blocking parallel code." Futures are a description of a task which will provide a result sometime in the future. Futures can be launched at any time and the result can be waited upon at a point where it is needed. This is similar to *threads* in parallel programming where the function launched by the thread would be the body of the *Future* and waiting on the result is analog to joining the thread.

Futures are complemented by *ExecutionContexts* which will handle the creation of threads, joining, and set the levels of parallelism, changing the *ExecutionContext* allows to adapt the global execution of concurrent code if needed. An more in-depth overview is available from the Scala documentation at :

`https://docs.scala-lang.org/overviews/core/futures.html`

6.4.4 Unit Testing

Some of the developed packages (scala objects) are accompanied by unit tests created to validate their correct behavior. These are specifications and tests that check if these specifications are met. Having unit tests allows for a higher level of confidence when making changes to existing functions.

Unit tests are also a way to check that important algorithms and computations exhibit the correct behavior. It is common for libraries as well as software to come with a collection of unit tests to validate their behavior. The unit test suite used in this project is ScalaTest, available at :

`http://www.scalatest.org/`

It provides a Domain Specific Language (DSL) to generate unit tests in an intelligible way. For example :

```
class ExampleSpec extends FlatSpec with Matchers {

    "A Stack" should "pop values in last-in-first-out order" in {
        val stack = new Stack[Int]
        stack.push(1)
        stack.push(2)
        stack.pop() should be (2)
        stack.pop() should be (1)
    }

    it should "throw NoSuchElementException if an empty stack is popped" in {
        val emptyStack = new Stack[Int]
        a [NoSuchElementException] should be thrownBy {
            emptyStack.pop()
        }
    }
}
```

And generates readable output messages when the test suite is run, when tests fail, they will also output readable (English) messages.

```
$ scala -cp scalatest_2.13-3.1.0.jar org.scalatest.run ExampleSpec
Discovery starting.
Discovery completed in 21 milliseconds.
Run starting. Expected test count is: 2
ExampleSpec:
A Stack
- should pop values in last-in-first-out order
- should throw NoSuchElementException if an empty stack is popped
Run completed in 76 milliseconds.
Total number of tests run: 2
Suites: completed 1, aborted 0
Tests: succeeded 2, failed 0, canceled 0, ignored 0, pending 0
All tests passed.
```

7 | Tools

Contents

7.1 Introduction

This chapter displays some tools that were created with the software framework developed during this projects. These are a few examples of what is made possible by the software library of packages created and give insights as to how these tools can be useful and could evolve into mature applications.

7.2 Signal Generator

The generic signal extraction and generation framework allowed to create a signal generator that would go through and entire alignment file (or part thereof) and generate signals that will be used by prediction and calling algorithms. The signals are also generated to be visualized to assess regions of the genome as well as possibly serve as inputs to other analysis programs or tools.

The signal generator uses the algorithm presented in section 4.2.1 to generate a signal on the genome. As described in the algorithms chapter, this takes an input alignment file, reads are filtered based on a given predicate and increments and decrements are generated for given positions given a function from which the signal is finally generated. Since this is very abstract and allows to generate any kind of signal from aligned input reads some examples are shown in this section. These examples will highlight the capabilities of the signal generator tool.

7.2.1 Signals chosen to be extracted by the tool

- **Read coverage** : This is the simplest example and shows the number of reads aligned to a given position.
- **Number of clipped reads** : The number of clipped reads, it shows the number of reads that are clipped overlapping the given region.
- **Number of reads with mapping quality 0** : This signal counts the number of reads that have been mapped to the position with a mapping quality score of 0 by the mapper/aligner. This score is given to show that the read maps to multiple positions on the reference.
- **Number of fragments spanning multiple chromosomes** : This signals counts the number of reads that have their mate mapped to another chromosome.
- **Number of fragments with discordant tandem orientation** : This signal counts the number of reads that are in a pair where both reads are oriented in the same direction (forward-forward, reverse-reverse).
- **Number of fragments with an inferred insert size that is too small** : This counts the number of reads that are mapped too close to their mate.
- **Number of fragments with an inferred insert size that is too big** : This counts the number of reads that are mapped too far away from their mates.

- **Sum of edit distance between reads and reference** : This is the sum of the number of differences between the read and the reference for all the reads that map to that position.

- **Number of reads that have their mate unmapped** : This counts the number of reads that have their mate unmapped.

- **Number of reads that have more than two elements in their CIGAR string** : The cigar string represents the alignment between the read and the reference (M is a match, I an insertion, D a deletion) so if for example a 150 bp read has a small 10bp insertion in the alignment its CIGAR string may resemble something like this "50M10I90M" and has 3 elements, a 50 base match, a 10 base insertion, and a 90 base match.

These are just a few possibilities, the list can be changed and new signal extractors can be easily generated with the tool.

Figure 7.1 shows the extracted example signals on a sub region of chromosome 7.

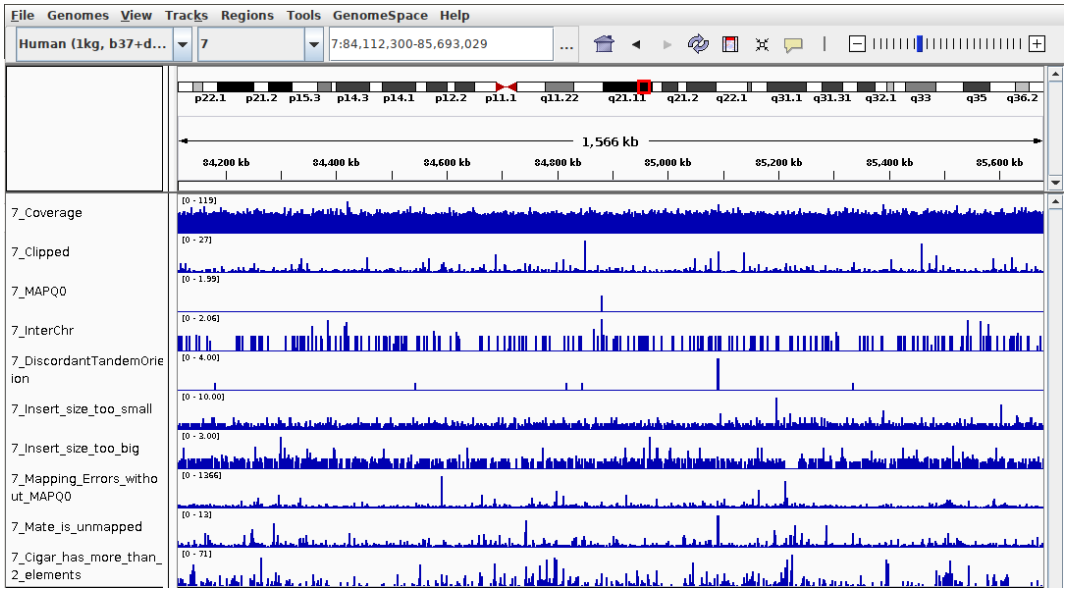


Figure 7.1 Extracted signals viewed in IGV

The morphology of these signals correlates with structural variation events and they can convey extra information for analysis or detection of structural events. A few examples of structural variation are shown below to display the relations between variations and signals. It is important to note that the scale of the signals is adapted to the current window (autoscale). The scale is shown on the top left corner of the signal window.

Insertions and deletions

Figure 7.2 shows the signals on a region that spots insertions and deletions. The signals are on the top, the short read sequencing reads that are at the origin of the signals are shown in the middle and at the bottom are long reads used as a reference to better see the variation events.

The first signal, the coverage (also visible above the reads in gray) shows a drop in the center were a deletion takes place. This deletion is on a single allele, therefore the coverage is not necessarily zero. This event is also marked by the second signal, the clipped reads. This is because the reads that span the deletion do not match the reference on one side, therefore they were clipped when aligned. The MAPQ0, inter-chromosomal breaks, and discordant tandem orientation signals do not show any activity on this region (the inter-chromosomal signal has 3 reads for this whole region that have their mate mapped to another chromosome, these events sometimes appear as artifacts from the mapping algorithm or sequencing errors in the reads resulting in incorrectly mapped reads). The "insert size too small" signal is also not relevant here. The "insert size too big" however has two peaks around the deletion event as to be expected, reads with an inferred fragment size that is too big are marked in red. Some of the reads around the deletion have their mate unmapped.

Finally the edit distance (mapping errors) signal and cigar element signal show peaks around insertion sites visible on both the short and the long reads. The clipped read signal is also active around these events as expected.

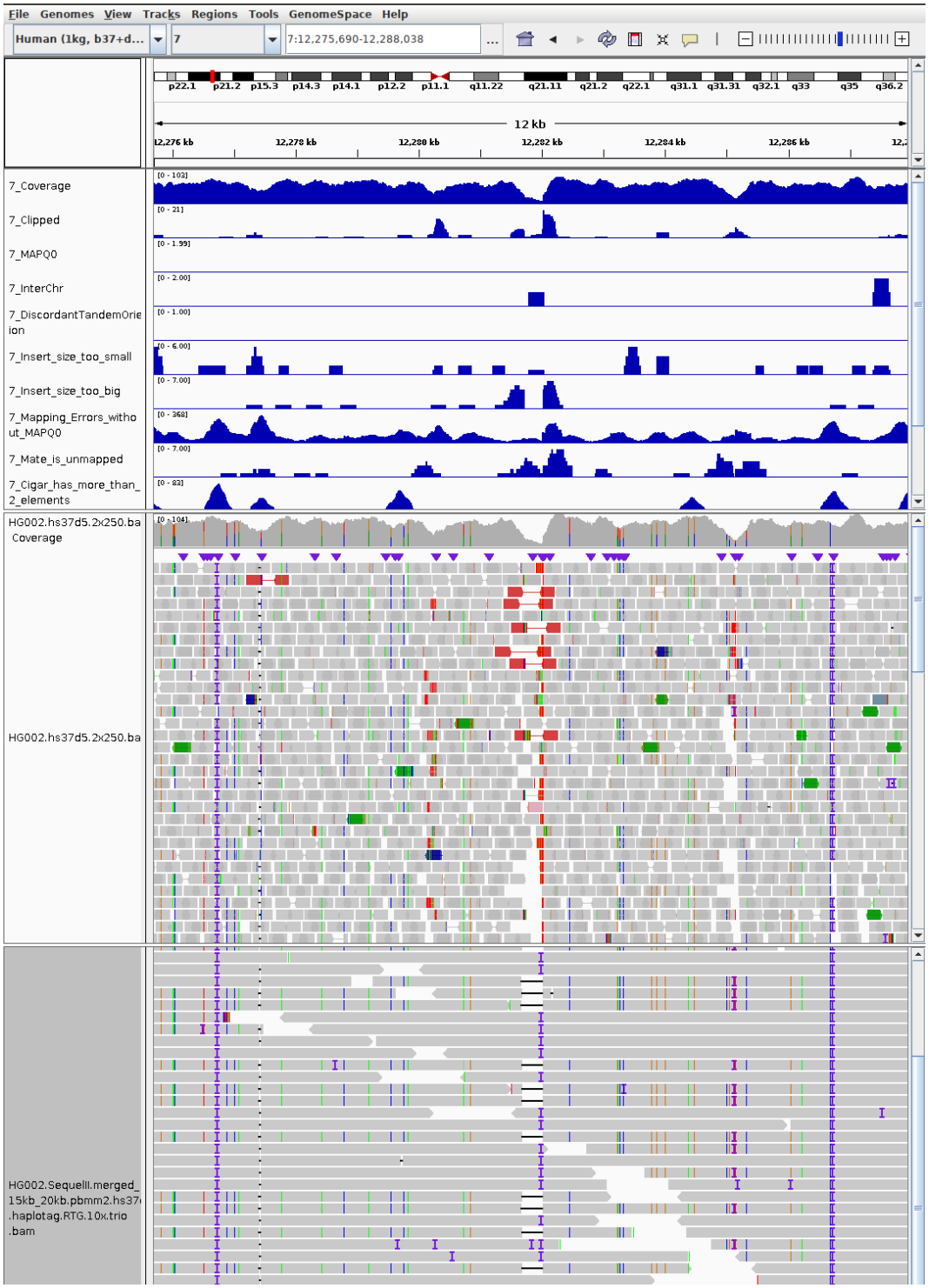


Figure 7.2 *Extracted signals viewed in IGV with short reads in the middle and long reads on the bottom*

Inversion

Figure 7.3 shows an inversion event and the extracted signals around this event. The event is clearly marked by the discordant pair orientation signal, the event also impacts the clipped read signal (around the breakpoints). On the same figure we can also see some small insertion and deletions marked by the signal that counts the number of elements in the CIGAR string of the read alignment.

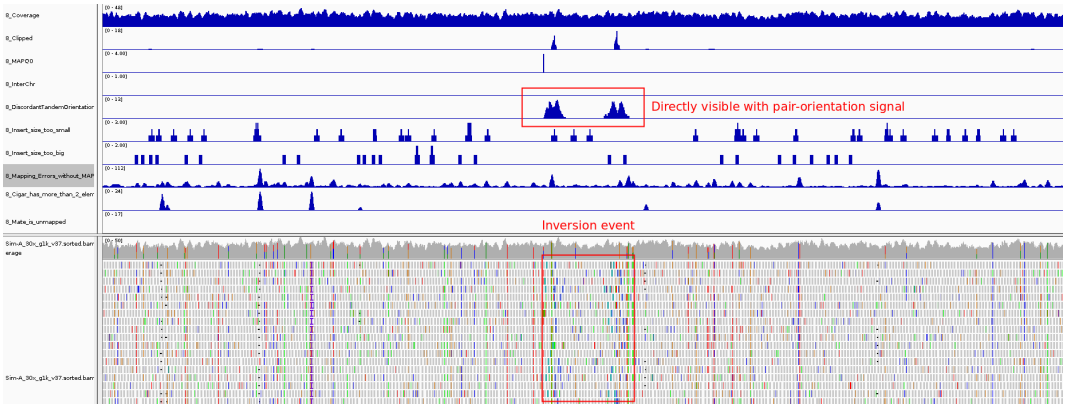


Figure 7.3 Extracted signals viewed in IGV around an inversion event

Complex event

Figures 7.4, 7.5, and 7.6 show a complex event and the values of the signals for the surrounding regions. The figures are views from an ever more un-zoomed perspective to show that the event region can still be seen even without the reads. The high levels of the clipped reads signal, edit distance (mapping errors), and unmapped mate signals are a clear giveaway of the event. The values clearly stand out around the event site compared to the surrounding regions (event always centered).

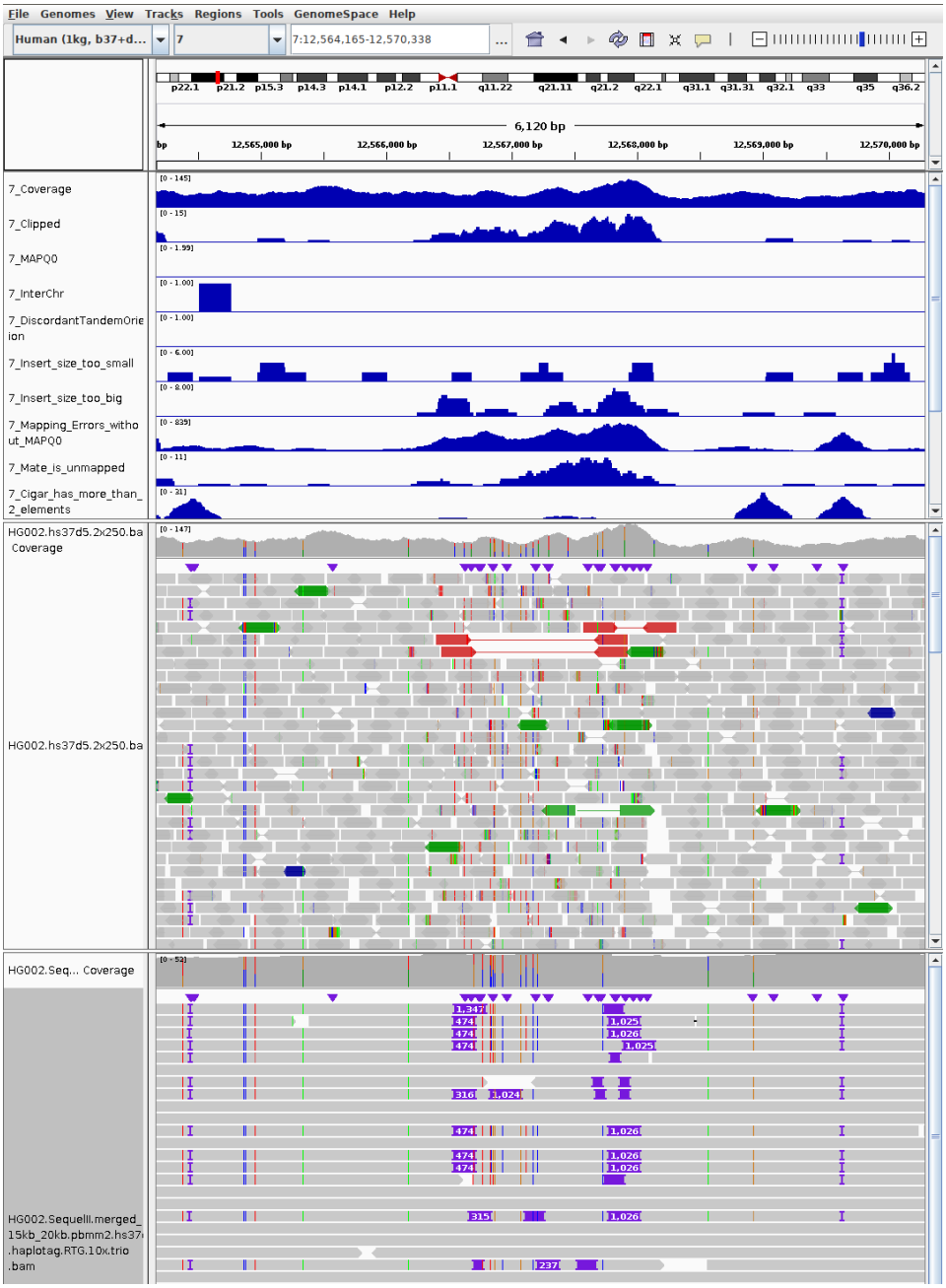


Figure 7.4 Extracted signals viewed in IGV around a complex event

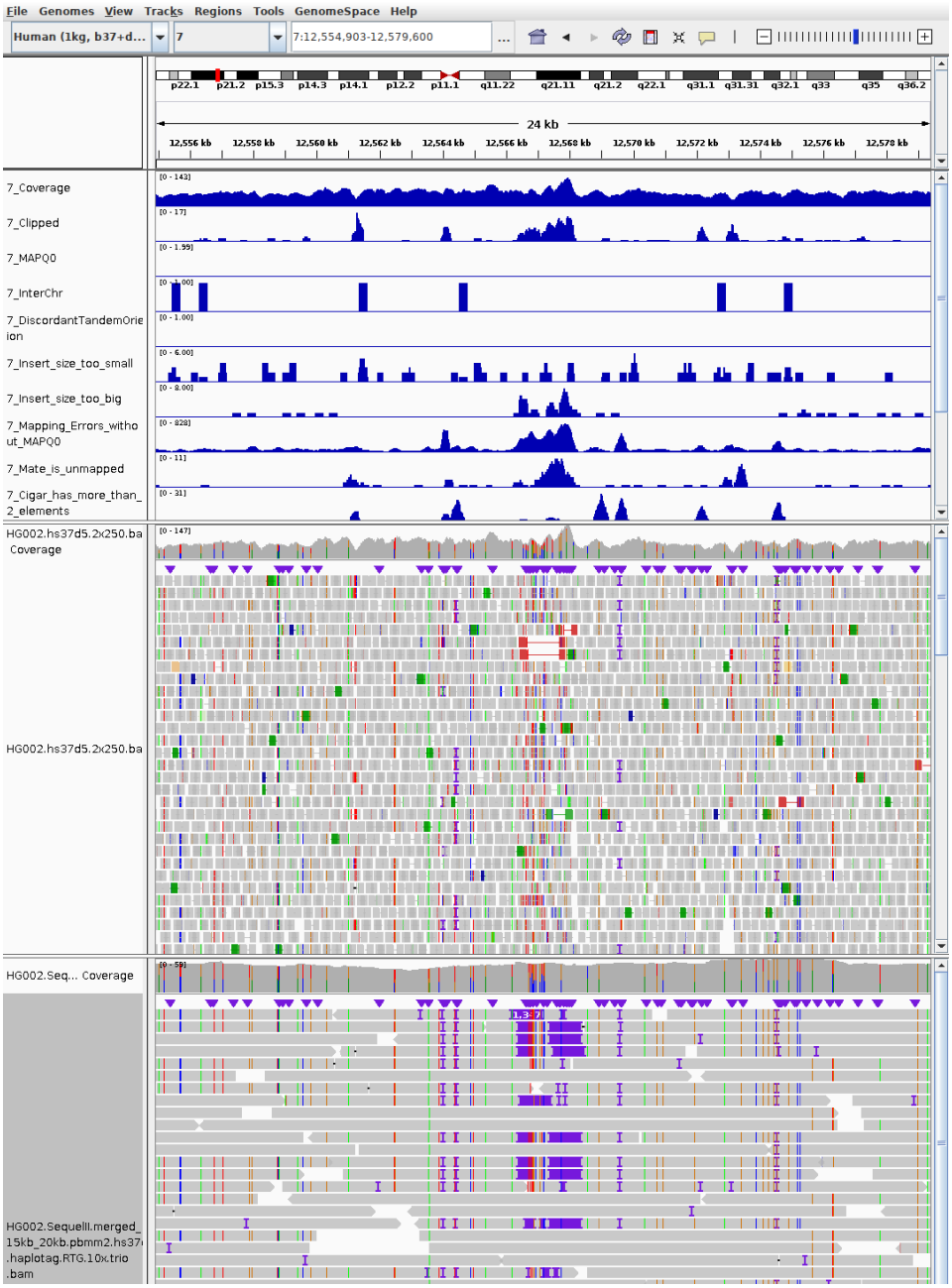


Figure 7.5 Extracted signals viewed in IGV around a complex event unzoomed

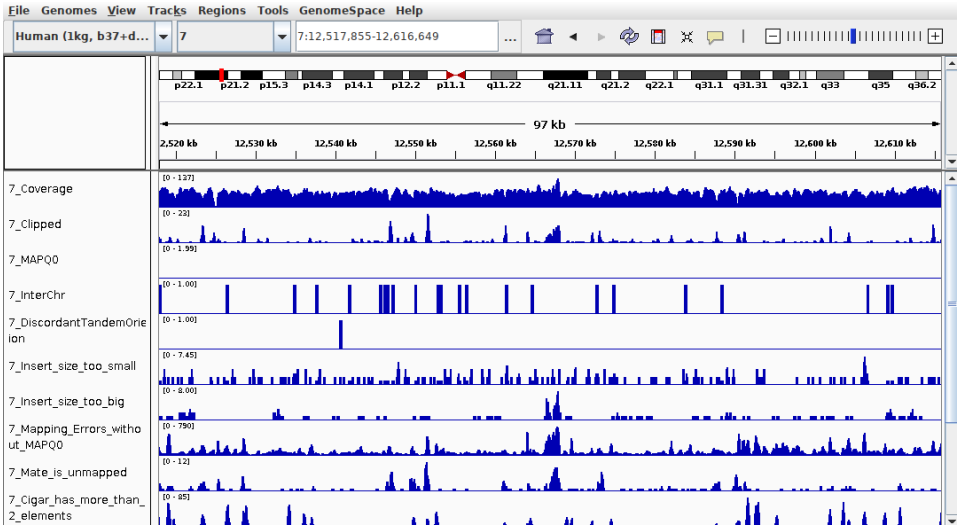


Figure 7.6 Extracted signals viewed in IGV around a complex event unzoomed more

7.2.2 Signal Processing

The signal generation framework not only allows to extract signals from an alignment file but also allows to process signals in order to generate new signals.

Possible processed signal examples include :

- Edge detection on coverage signal. This signal allows to detect abrupt changes in read depth. This signal correlates almost directly with deletion and duplication variation breakpoints, examples are shown in figures 7.7 and 7.8

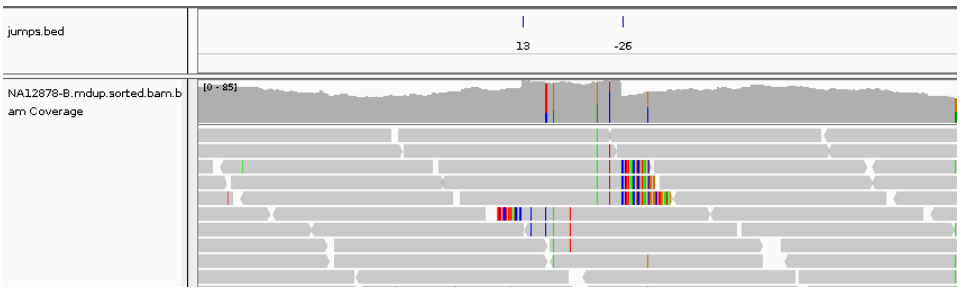


Figure 7.7 Edge detection signal on coverage with a threshold of 8

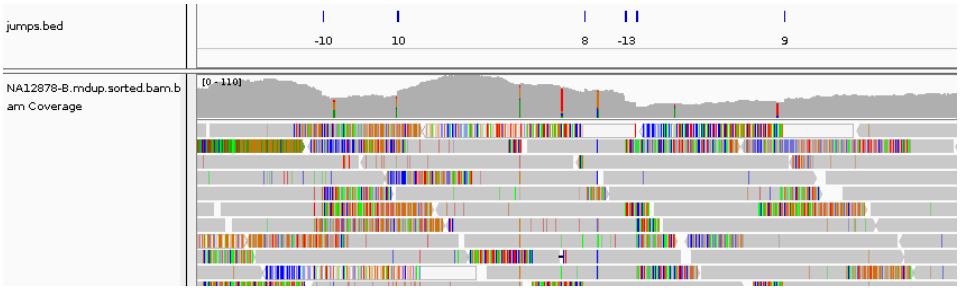


Figure 7.8 Edge detection signal on coverage with a threshold of 8 on a complex region

- Low pass filtered coverage binned by sequencing depth. This signal is a direct approximation of the copy number of any region of the reference. An example signal is shown in figure 7.9.

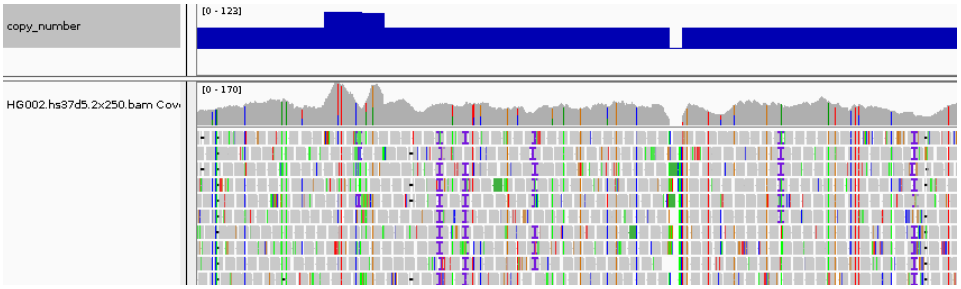


Figure 7.9 Segmented, low-pass filtered coverage signal, indicator of copy number

Possible combined signal examples include :

- Signals normalized by the coverage (read depth, read count) signal. For example normalizing the edge detection signal of the coverage by the coverage itself allows to filter some false positive edges that occur in highly covered areas, where the jumps in coverage are high compared to the average sequencing depth but small relative to the local coverage.

- Subtractions between signals, for example subtracting the number of reads with low quality mapping or non unique mapping to the coverage signal allows to generate an alternate coverage signal that may be a better lower bound to the actual coverage, because reads that map to multiple locations may artificially increase the coverage if they are counted. Figure 7.10 shows two coverage signals, one with reads of mapping quality 0 and the other without.

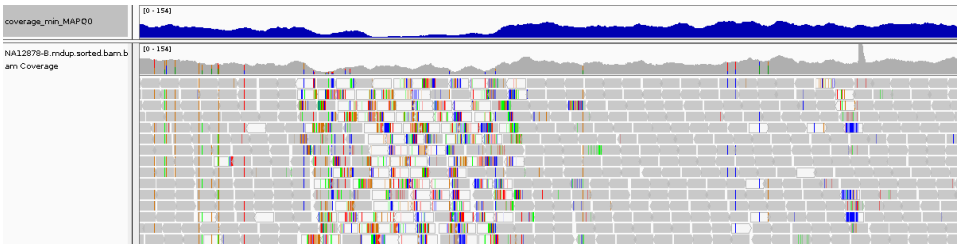


Figure 7.10 Coverage signal minus reads with mapping quality 0

Note : The coverage generated by IGV can be lower at some points compared to the blue generated coverage because the generated coverage did include clipped ends of reads where IGV did only account for the unclipped portion of the read. This is an arbitrary choice and can be changed in the specific signal generator. There are many possibilities to generate the coverage signal, taking the whole read into account, only the unclipped part, but it is also possible to adapt the coverage more by interpreting the exact alignment of the read and decreasing the coverage based on deletion in the alignment. These choices depend on the needs of subsequent analysis and can be easily changed in the signal generation tool based on need.

7.2.3 Conclusion on the signal generation tool

The generation of the signals is necessary for the subsequent detection of CNV events, but the signals can also help to visualize properties of the aligned reads that are not clearly visible with the reads alone. They also help to assess those properties on bigger regions and give a more global idea of the region. These signals can also guide manual exploration of the genome and serve as input to other analysis programs.

7.3 Predictors

The generic prediction framework allows to takes input signals and a function to combine these signals as well as a function that uses these signals to call variants from the input data. Examples of created predictors include :

- A predictor that gives all regions that have an unexpectedly low coverage as shown in figure 7.11. This is a very simple predictor that simply filters the coverage signal and outputs regions where it is below a threshold. It can serve as a base for a deletion caller.

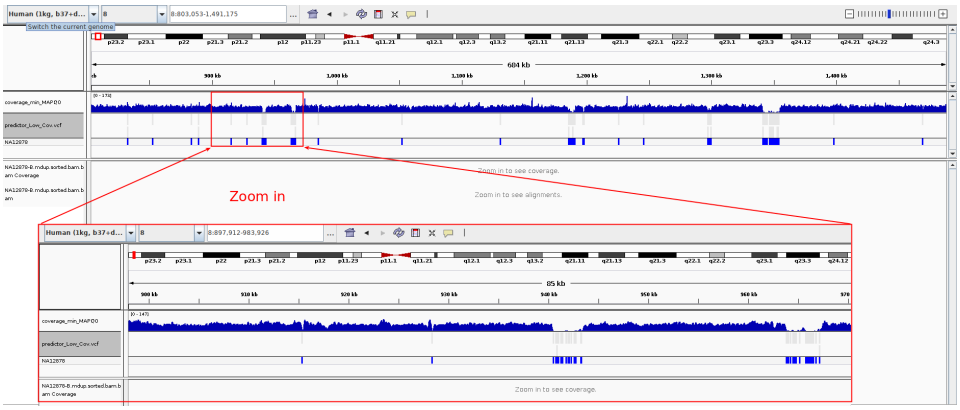


Figure 7.11 Low coverage predictor.

- A predictor of breakpoints based on edges in the coverage signal, this is based on the edge detection signal introduced above and can serve as a predictor for SV breakpoints in general.

- A predictor of breakpoints based on edges in the clipped read signal, similar to the previous one, it can serve as a predictor for edges of SV events in general.

- A duplication predictor based on copy number from a segmented filtered coverage signal, figure 7.12 shows an example of a predicted (detected) duplication of more than 100k bases on NA12878. The same signal is shown in figure 7.13 where a deletion on a single allele of more than 150k bases is visible in the signal.

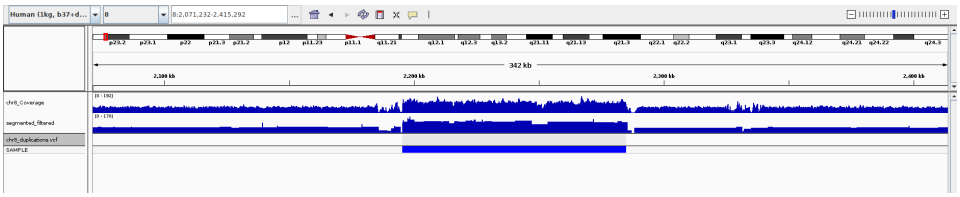


Figure 7.12 Predicted duplication on NA12878

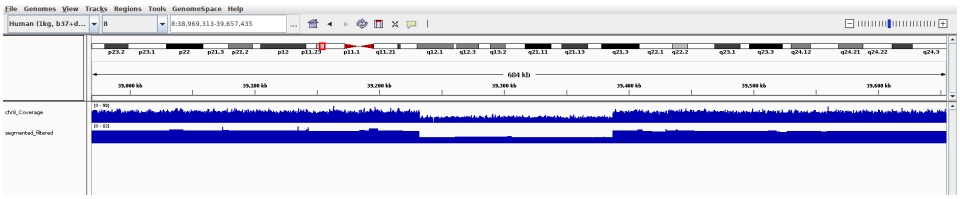


Figure 7.13 Single allele deletion of more than 150k bases on NA12878

- The generic predictor that can take any signals and a generic prediction funtion given theses signals to create any predictor one can think of.

These are a few examples and the predictors should be adapted for each task, the idea was to give a very generic prediction framework in order to be able to experiment and be able to rapidly create any predictor needed by a given calling algorithm.

Predictors can also be used on their own to generate output files (BED or VCF) to indicate predicted regions on the genome given a specific property.

7.4 Variant Callers

Combining the prediction framework and algorithms for analysis and calling made it possible to create variant callers easily and effectively. These include :

- A deletion caller based on inferred fragment sizes
- An inversion caller based on read-pair orientation
- A duplication caller based on copy number estimation from coverage
- An interchromosomal event caller based on read-pairs
- An insertion caller based on clipped reads

These variant callers are examples and are not specifically tuned or optimized. They are evaluated in chapter 8 (results) against the state of the art and examples will be given in that chapter. Because they are discussed and reviewed in detail in chapter 8 no further details or examples are given in this chapter.

7.5 De-Novo Assembler

A De-Novo assembler was developed during this project in order to solve some of the detected variants. It is an augmented de Bruijn Graph based assembler. The algorithms related to this assembled are described in section 4.5

The resulting assembler is a collection of tools that can be run either on its own in a "best-effort" mode were the tool tries to solve given regions as best as possible or run in an interactive Scala console to explore assembly of given regions. The interactive mode allows to create the augmented De Bruijn Graph, apply algorithms on it, display the graph with Gephi, extract possible contigs and re-align them back to the reference. The "best-effort" mode will simply try assembly on a list of given regions (e.g., all predicted insertion sites) and generate alignments of the biggest contigs it could generate back onto the reference.

7.5.1 Assembly examples

Below are few examples of assemblies and realigned contigs generated by the tool.

Deletion on both alleles

This is a simple example to show the resolution of a deletion case with local assembly. Figure 7.14 shows sequencing reads at the top, extracted reads around the event for local assembly in the middle followed by the assembled contig and finally, long reads at the bottom for reference (not used in assembly).

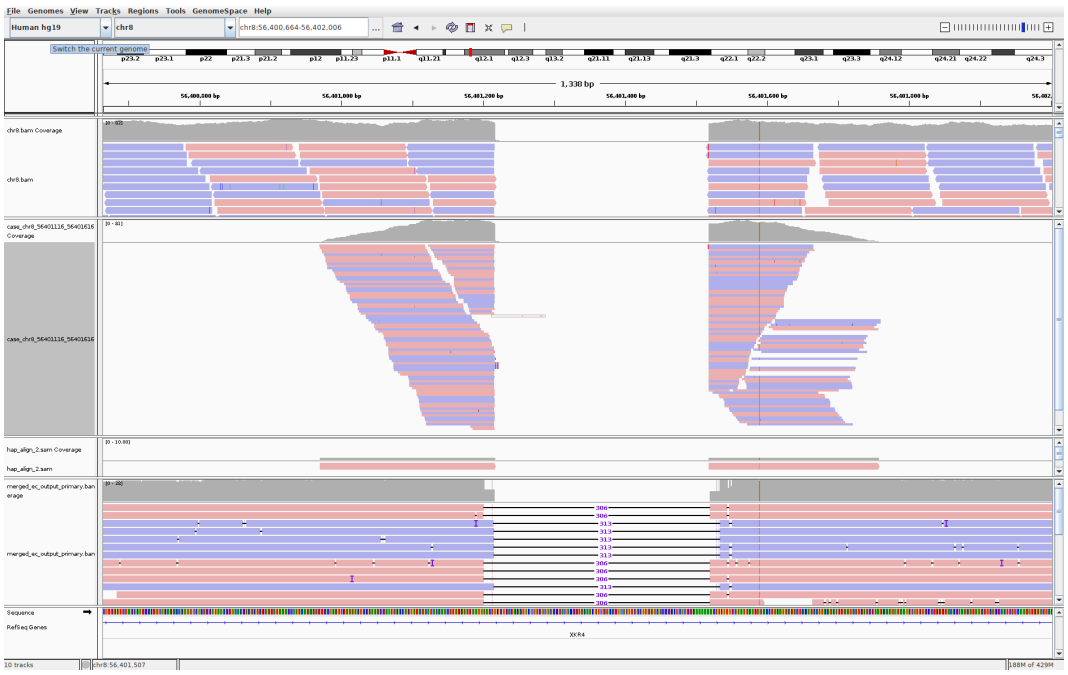


Figure 7.14 Deletion solved with local assembly

The first assembly graph generated by these reads is visible in figure 7.15. The size of the k-mers used to generate the graph is 50. We can see that the graph has bubbles, many "hair-like" tails, and is split into to disjoint graphs.

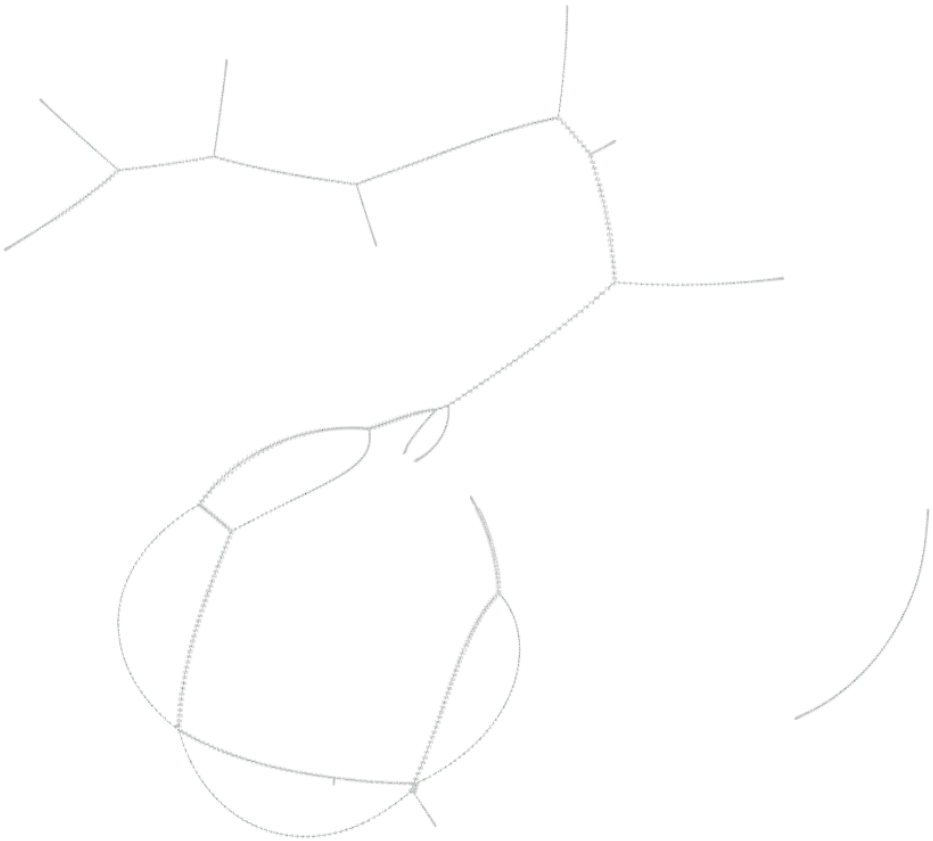
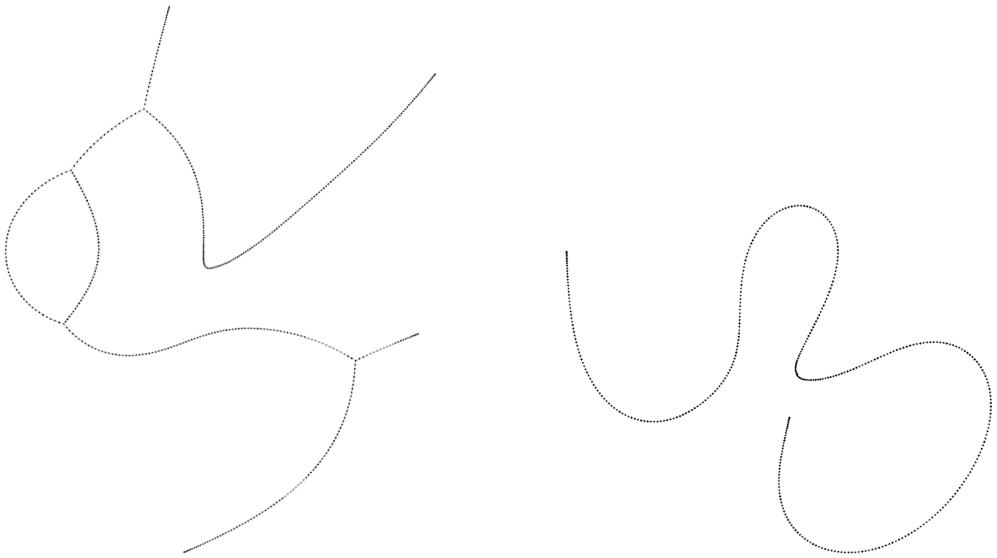


Figure 7.15 Initial De Bruijn graph of a deletion with k-mers of size 50

The rightmost graph (single strand) is actually a mismapped read with a mapping quality value of 0. It is visible in the middle of figure 7.14 in a lighter color. When filtering the graph by the weight of the edges, removing all edges of weight 1 (single occurring overlap) we achieve the graph shown in figure 7.16a. The bubble comes from two reads that exhibit the same error on the same base, therefore the edges on one side of the bubble are of weight 2. If we continue and remove all edges with weight 2 we achieve the graph of fig 7.16b.



(a) De Bruijn graph of a deletion with k-mers of size 50, edges of weight 1 removed

(b) De Bruijn graph of a deletion with k-mers of size 50, edges of weight 2 removed

Figure 7.16 Pruning of the assembly graph based on edge weight

This graph is a single contig and is therefore solved. It can be realigned back to the reference based on the position of the initial reads that generate the assembly, the positions can be inferred from the two ends that came from reads that map correctly and the deletion gap can be obtained by realignment to the reference with the Needleman-Wunsch algorithm. The resulting contig realigned to the reference was shown in figure 7.14.

This is a simple case that can be solved without assembly but provides a clear view of the assembly graph and how removing the edges with too little evidence makes the case clear.

Insertions and SNP example

Local assembly is especially useful to solve insertion cases, where novel content is found in the individual genome compared to the reference. Due to the short sequencing read length only small cases can be solved. Figure 7.17 shows the resolution of a region with two insertions around a single SNP. The short reads are on top, then long reads as a reference (not used in assembly) and finally the generated contig realigned at the bottom exhibiting the two insertions around a single SNP.

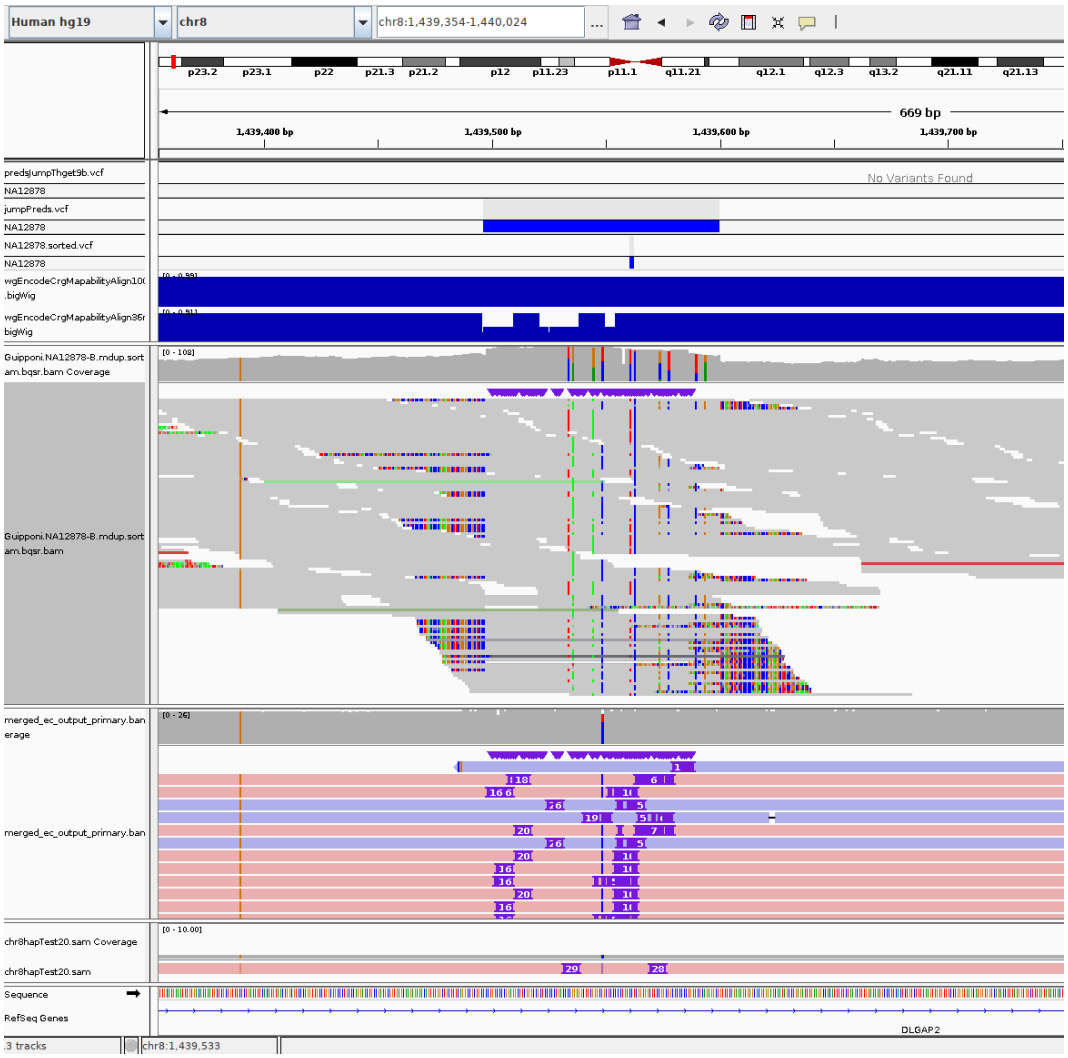
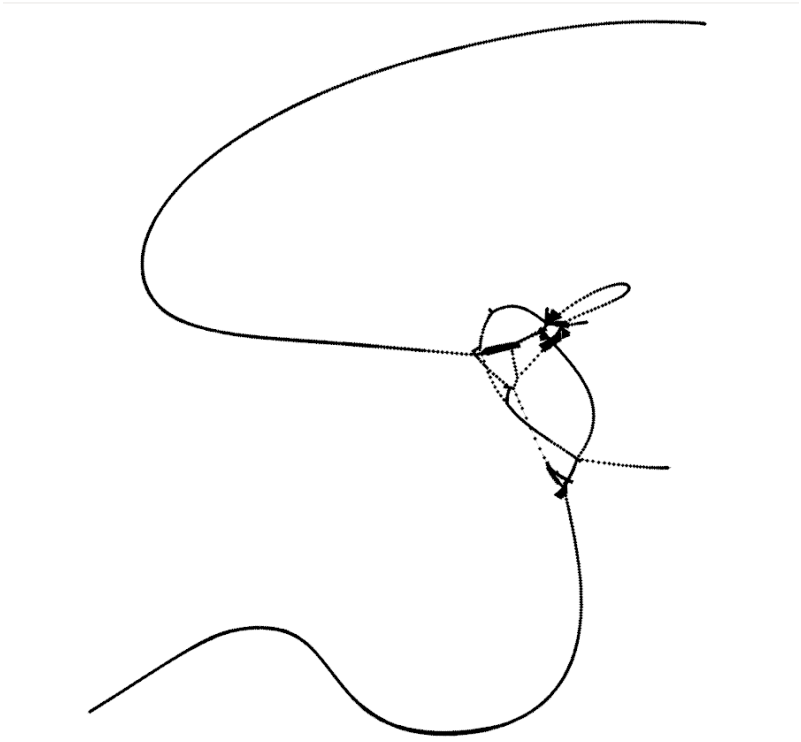
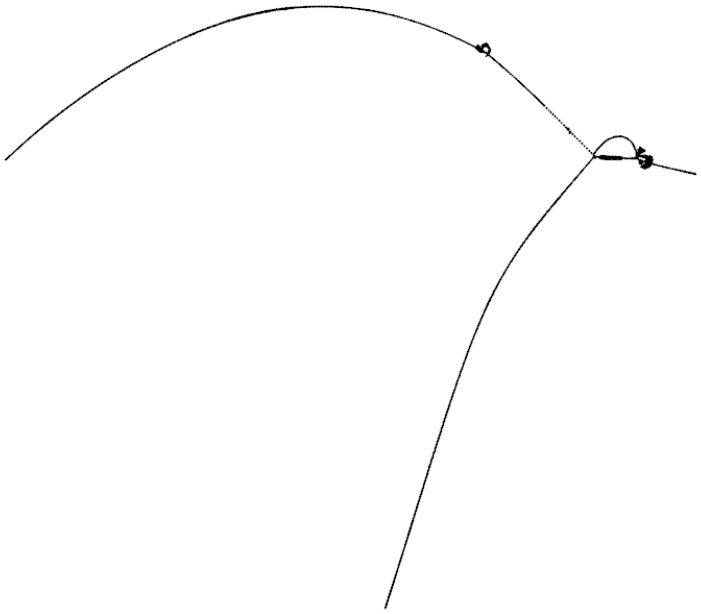


Figure 7.17 Resolution of two insertions around a SNP with local assembly

Figures 7.18a and 7.18b show the assembly graph being pruned. Pruning was iterated until a single contig remained. This is not always possible, for example with repetitions the graph will have loops with possibly high weight edges and other approaches must be taken. For example mapping both ends around the loops and estimating the copy number of the duplications based on the total edge weight compared to the average edge weight.



(a) De Bruijn graph of a region with insertions with k-mers of size 50, edges of weight less than 10 removed

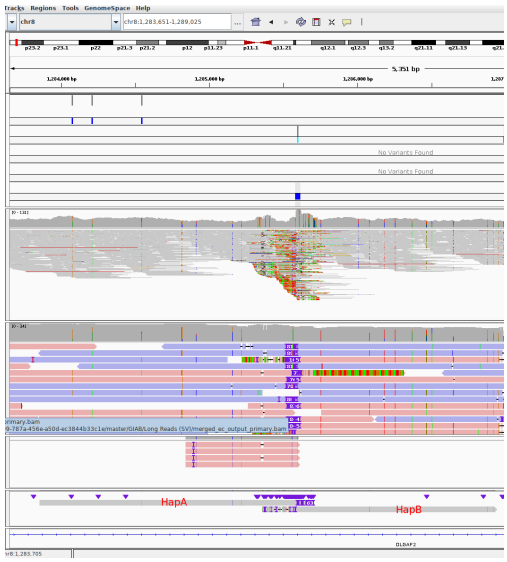


(b) De Bruijn graph of a region with insertions with k-mers of size 50, edges of weight less than 15 removed

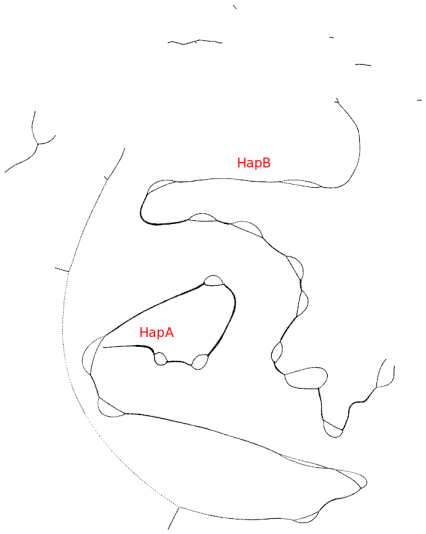
Figure 7.18 Pruning of an assembly graph based on edge weight

Complex duplication example

As a last example a complex duplication event is shown, this example could not be solved by local assembly but partial resolution was possible. Figures 7.19a and 7.19b show a complex region and the assembly graph side-by-side. Two haplotypes (haploid genotypes) contigs were visible but are disjoint because the novel content joining them is missing (unmapped reads). Bubbles in the graph are when paternal and maternal alleles differ and come from SNPs here. The bubbles in the graph come from the diploid sequences in the genome. Each graph was traversed by picking a random path for each bubble (not phased) and aligned back to the reference, we clearly see the edges of the event and some of the novel insertion in the middle, from both sides, but not all of it.



(a) Complex region with duplications and novel insertions



(b) De Bruijn graph of a complex region with insertions and duplications with k-mers of size 50

Figure 7.19 Pruning of an assembly graph based on edge weight

A final attempt at solving this case was to add the unmapped mates of the reads during the assembly. This resulted in the graph of figure 7.20. This did create bigger contigs for each side but did not join them. A duplication loop also appeared at the bottom of the right assembly. This case is not easily solvable with short reads, if solvable at all. In order to attempt to solve this case a more global assembly must be done, meaning that other sequencing reads must be added to the graph, because the missing link (missing reads) may be unmapped or simply mapped to the wrong region of the genome. Here we are sure some reads are missing because the two assemblies (left-side, right-side) are disjoint.

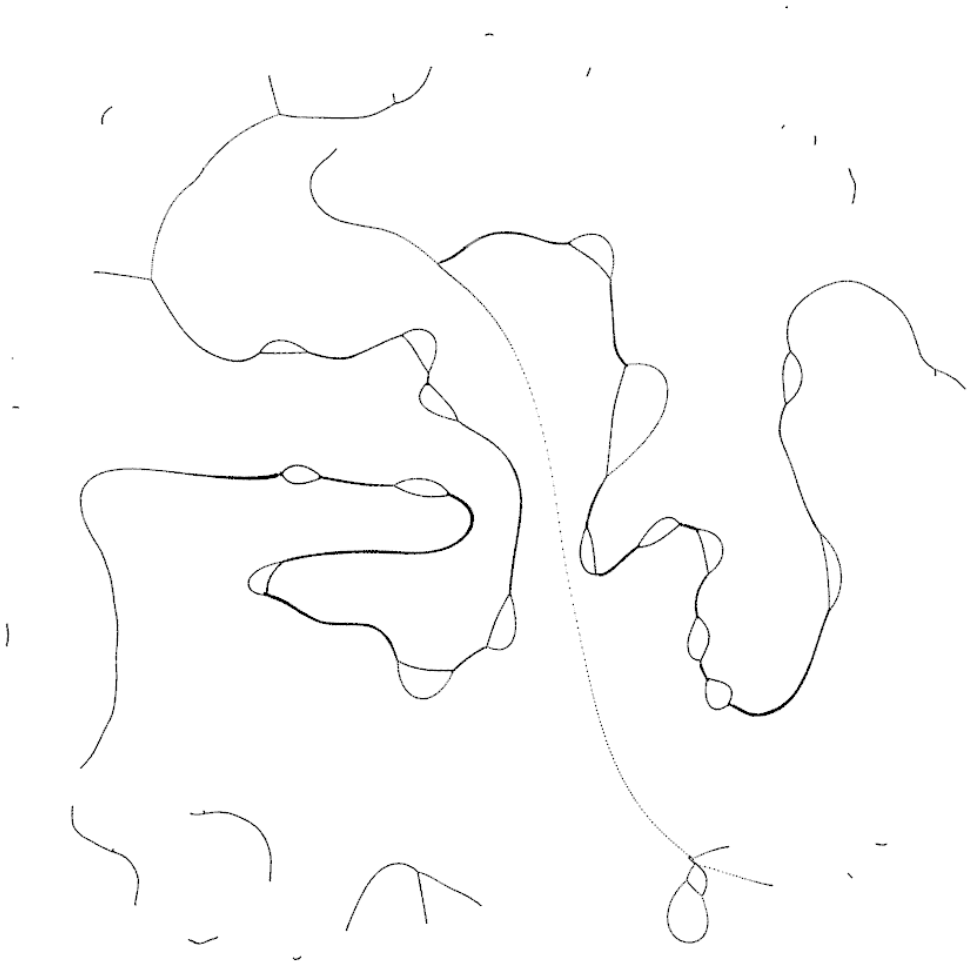


Figure 7.20 De Bruijn Graph of complex event with unmpapped mate reads

Further examples of generated contigs

Figures 7.21 to 7.22 show example contigs aligned to the reference. Shown with short reads (that were used for the assembly) on top and long reads at the bottom to serve as a reference for comparison. Some assemblies were validated by other studies because the variants for NA12878 were found in variant databases, no in-depth evaluation of the assembled contigs was done because of time constraints. The contigs are also unphased and therefore variants on a contig may not necessarily be on the same allele in reality. However, the generated assembly graphs and their extra properties enable subsequent in-depth analysis if needed. This would not be possible with the standard de Bruijn Graph approaches because they only keep k-mer information and therefore loose information about the provenance of the k-mers and their relation to the reads.

Figure 7.21 shows three contigs that were generated with the assembly graph. They all show an insertion around the variation event. This insertion is also visible on the long reads (on the left for forward reads and the right on reverse reads because of the alignment). This insertion is a small difference of 10 extra "T" bases in the individual genome compared to the reference on a repetitive "TTTT..." region.

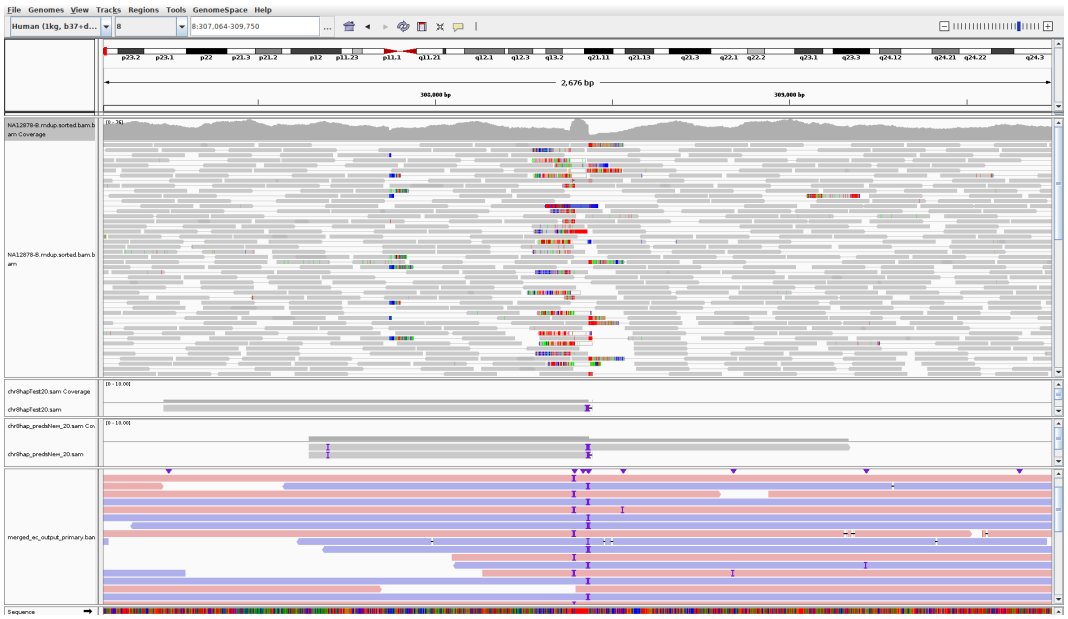


Figure 7.21 Small insertion

Figure 7.22 shows two predicted insertion sequences of 60 and 49 bases. Long reads seem to indicate that the insertions are on a single allele only. A structural variation study based on long reads found this insertion site with local assembly and predicted a single 43 base insertion. The data is available under :

`ftp://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/data/NA12878/NA12878_PacBio_MtSinai/`

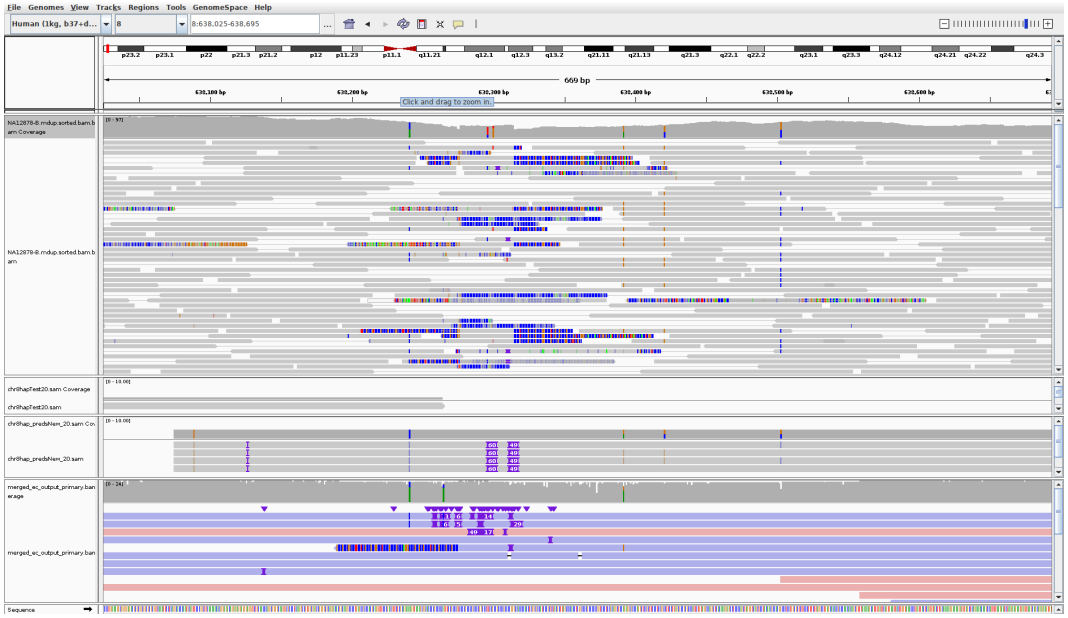


Figure 7.22 Insertion on single allele

Figure 7.23 shows a 33 base deletion solved with the local assembly algorithm. This 33 deletion is also confirmed by the same structural variation study linked above using local assembly on long reads. This deletion is known and available in the small variant truth set for platinum genomes that can be found here :

`https://hgdownload.soe.ucsc.edu/gbdb/hg19/platinumGenomes/`

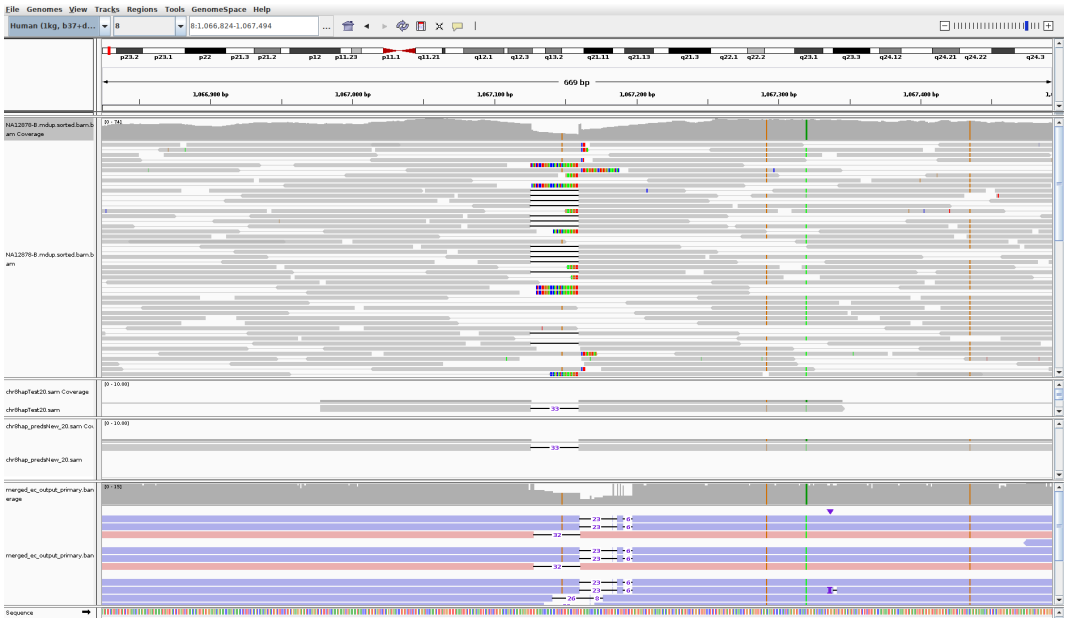


Figure 7.23 Solved homozygous deletion event

Figure 7.24 Shows a complex event that is partially solved, contigs were generated by traversing the loops in the graph only once and are therefore not correct. This event was found by other studies and other software such as PBHoney on long reads and marked as a 278 base insertion. Entries for duplications and insertions around this region can be found in the database of genomic variation.

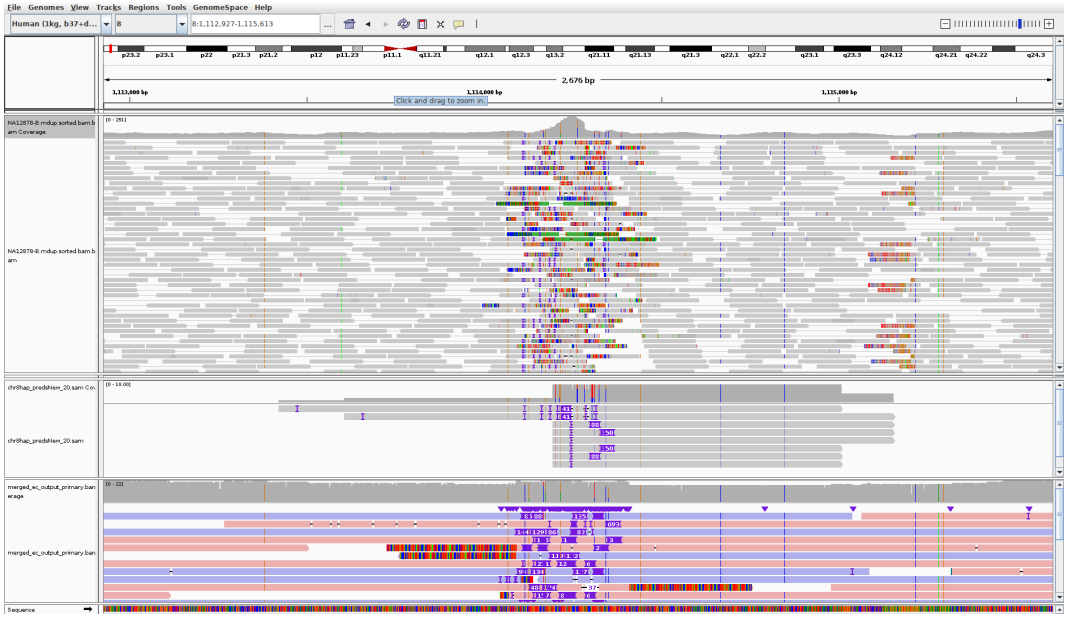


Figure 7.24 Complex event, partially solved

Figures 7.25 and 7.26 show insertions found with the assembly algorithm, it shows that the graph is able to generate contigs for both homozygous and heterozygous insertions.

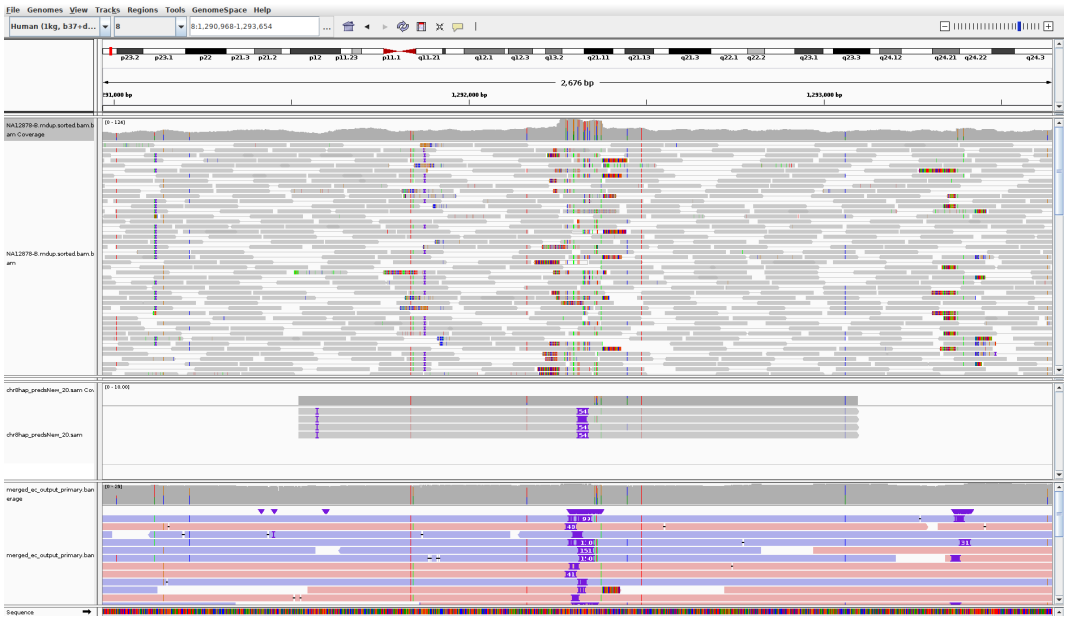


Figure 7.25 Homozygous insertion (both alleles)

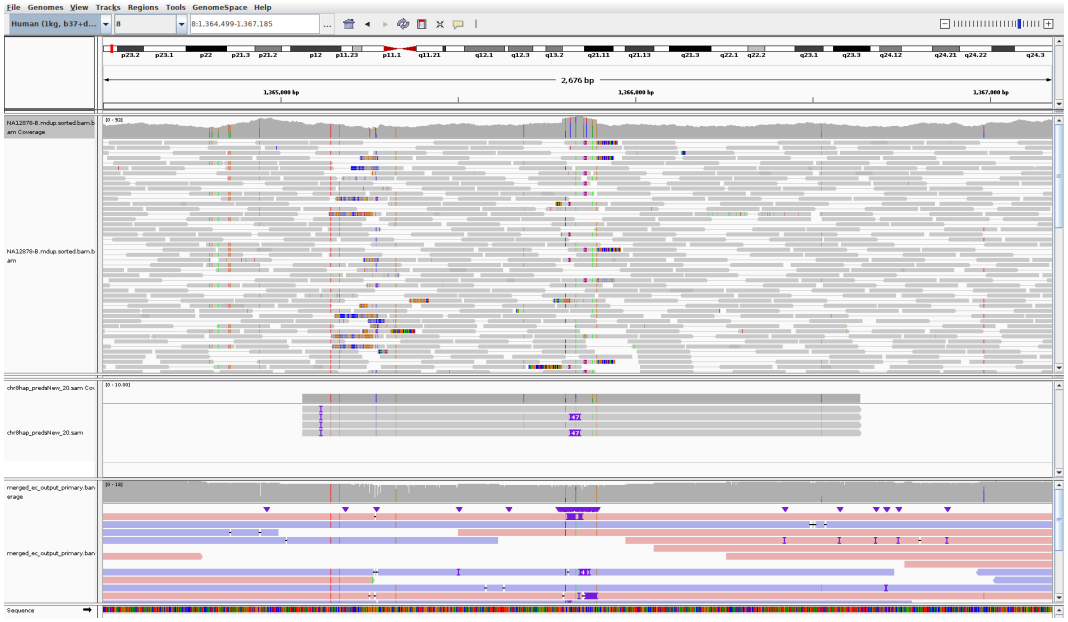


Figure 7.26 Heterozygous insertion (single allele)

Figure 7.27 shows an insertion that is referenced in dbVar as nsv396086 and found here with local assembly. This concludes the small list of examples for the assembly tool.

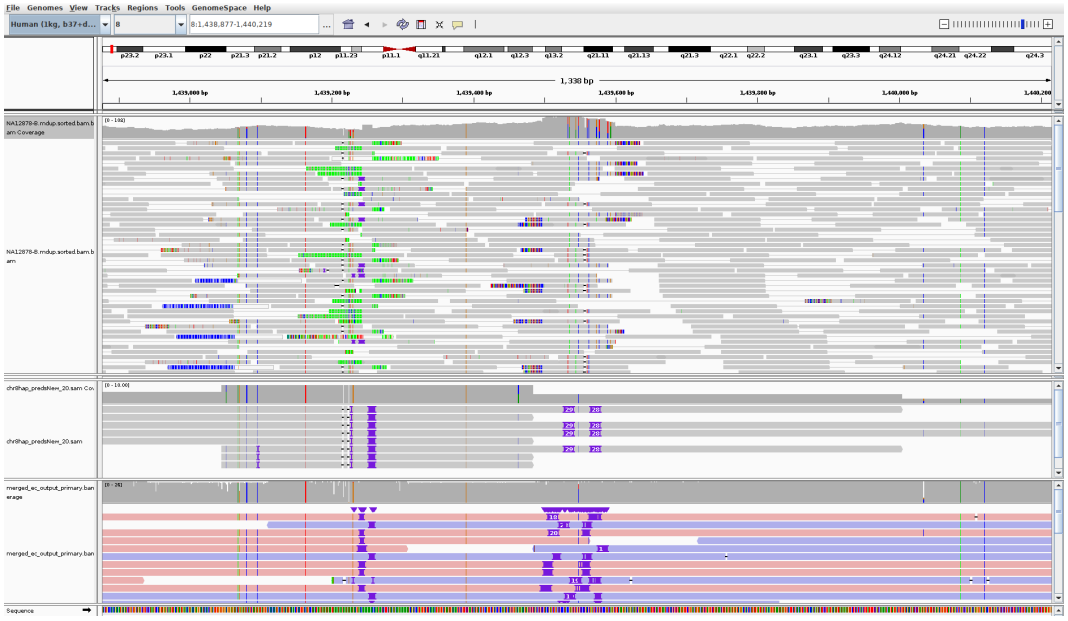


Figure 7.27 This example is interesting since the two insertions around a SNP correspond to the nsv396086 entry found in dbVar

7.5.2 Conclusion on the assembly tool

The assembly tool is capable of solving some variation events. Deletion events and small insertions (less than a fragment in size) can often be solved. The more complex events can be partially solved and some events cannot be solved at all. However this assembler is still a valuable tool and may help to visualize and assess complex events complementing the sequencing reads. For example, the unsolved cases could be manually assessed and explored with the interactive mode.

The tool can be run automatically in a best effort way to solve detected events but the real value is in the potential to interactively explore variant sites, building and displaying the graphs, trying filtering algorithms, not only on edge weight but with any of the meta properties of the augmented de Bruijn graph. Exploratory possibilities include manually choosing to add reads from the alignment file e.g., based on closeness of mates of the reads used in the assembly or adding unmapped reads with certain properties and observing the results. Trying to apply filtering based on meta-properties can also help explore complex cases.

De-novo assembly is a very broad topic in itself and this tool was created to explore concepts and strategies that could be applied to create new assembly techniques or heuristics for variant calling. For example, duplications can be assessed from the presence of loops in the graph and their copy number can be estimated from the coverage. The main benefit of assembly remains its ability to provide the sequence for insertions of novel DNA content, which with short reads is often only possible for short insertions.

7.6 VCFotographer a Structural Variation "Photobook" generator

VCFotographer was developed as an automated variation capture tool. Mimicking what a researcher would do with a variant collection. Opening the file in IGV, going to the locations of interest and looking at the sequencing data (and in our case signals). VCFotographer is a fully automated tool that can take a collection of variant calls as input and generate a full "photobook" with screenshots of all the variants shown in IGV with any number of sequencing reads and data tracks.

VCFotographer is completely system agnostic (it runs in a docker container) and does not require a graphical output (it can run in the cloud or on a server). It accepts VCF files for the input variant collection and any other IGV supported file for the extra displayed tracks. It outputs screenshots of IGV in png format with the chromosome name and coordinates of the events as well as all the tracks.

The output screenshots could then be further incorporated in the automatic generation of a PDF report file for the given variant collection.

```
rick@A13PC04:/media/rick/extra_space/Master/master-hes-so/docker/VCFotographer$ docker run
↪  rwe/vcfotographer vcfotographer

                IMPRECISE;SVTYPE                  VCFotographer
     FILTER   length,assembly=hg19
   QUAL   BL  PQ                fileDate=2020      Author : Rick Wertenbroek
  INFO      1KGP    SVLEN=+42;              FILE
  ID              REF       PASS      VCF     ZU   If you use VCFotographer
                BQ   0/1      PASS                 in your research please
              SB   ok           PASS               cite :
             AA  cG               del
            DB  :O                 ins             TODO CITATION
            MQ  .                  dup
            GL                     inv
             MQ                   cnv
              ALT                bnd
                ook             PASS
                  END       PASS
  NS                VALIDATED'                GT
  GCCACNACGCCTGGCTAATT-----TATTTTTACTAGAGACGGGGT
    TATAAAACNANACTTCAGAATTACCATAATATTGATTACAAT


14:30:47.370 [main] INFO Log - VCFotographer version 0.1.0-SNAPSHOT
Usage: vcfotographer -i variant.vcf -b reads.bam [--scaling factor] [-a additional_track] [--output-dir
↪  output_directory]

    Additional tracks - Can be of any type supported by IGV, big file may slow IGV down
```

Figure 7.28 shows a screenshot of the output of VCFotographer, a photobook (collection of screenshots) of every variant from the input variant file in IGV. The files are named by region and position of the event. The screenshot shows the capture of a deletion event on chromosome 8 with aligned sequencing reads.

The creation of VCFotographer emerged from a small tool that was used throughout the time of this thesis. A tool that could send commands to IGV through a network port. This made it possible to control IGV and display any region upon request. This was used to look at variants, debug cases, and display any region of interest, therefore making an automated program to capture variant calls was a natural extension of this tool.

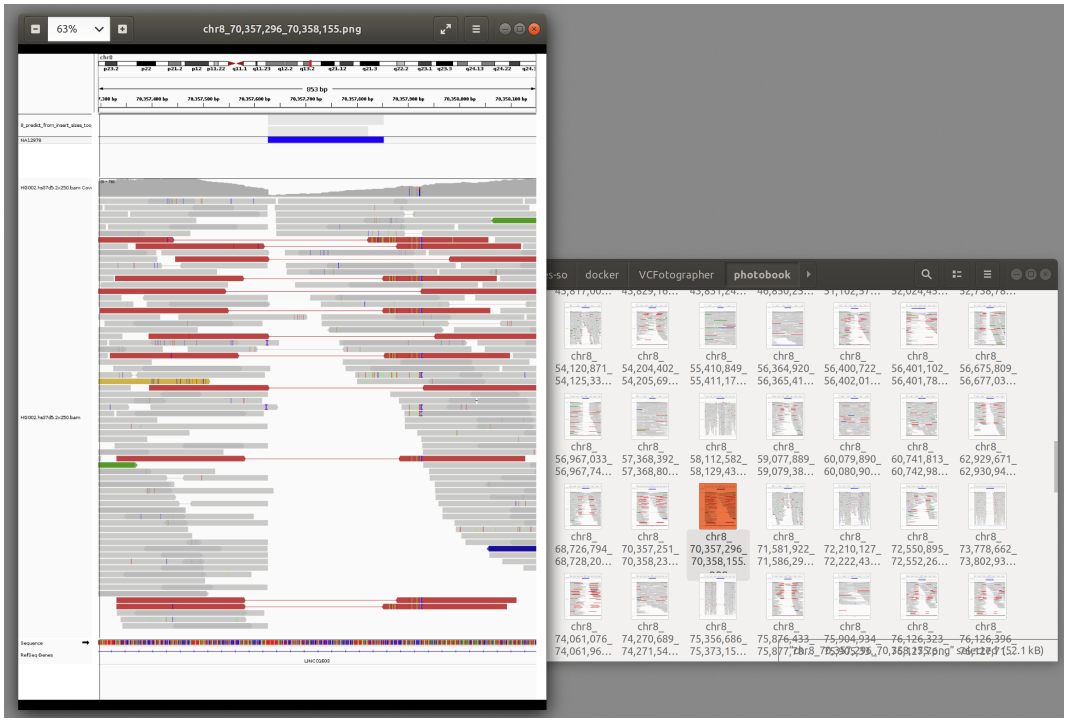


Figure 7.28 Screenshot of a photobook generated by VCFotographer

8 Results and Evaluation

Contents

8.1 Introduction

This section will evaluate some of the variant callers created with the tools and software framework of this project. They will be compared in performance against the state of the art. The benchmark of their detection capabilities was done on both real sequencing samples and simulated data. The results will be explored as thoroughly as possible. However, given the very generic and modular nature of the framework it will not be able to explore the endless combinations and calling methods made possible by the framework. Calling algorithms benchmarked are presented with a small discussion of their algorithm and parameters. The impact of the algorithm and given parameters is discussed with some selected examples. This exploration of the results will give insights as to how different parameters or choices in the algorithms do influence the calling results. This will also give insights as how to improve the variant callers.

8.2 Simulated Dataset

A simulation dataset was used in order to have a ground truth of variants tu be used for detection. The dataset chosen is the same one as the one used in the benchmarks [56], [190]. Their study evaluated 69 calling softwares on this dataset and provide all the resulting calls. This makes it possible to compare the results of this project against existing software without having to rerun every experiment. This also allows the reader to further compare the results with all the 69 calling softwares evaluated since only the best few are shown here for comparison.

The simulation dataset was recreated following the instructions of this study. The simulated structural variations (3,530 DELs, 1,656 DUPs, 2,819 INSs, and 309 INVs of which 4,853 are heterozygous SVs) have been inserted into a diploid reference sequence and then artificially sequenced using the ART sequencing simulator software as paired-end reads of 125 base pairs and an average insert size of 500 base pairs with a standard deviation of 100 bp. The Illumina HiSeq2500 profile was used to simulate sequencing technology errors and properties.

The artificially sequenced reads were then realigned to hg19 using BWA version 0.7.17 and this provided the BAM file that was used for the benchmark. The details and exact instructions to recreate this dataset are available in chapter 10 Materials and Methods.

8.2.1 Results

Signal generation has been run on the entire simulated genome. Time, memory, and disk space considerations are given in section 8.4.

Three different callers created with the software developed within this project have been evaluated on this data set. One for deletions, one for duplications, and one for inversions.

Calls are evaluated true positives (correct) when they have at least a 80% reciprocal overlap with the actual region of the event. This is the same criterion used in the study that provided the results for the other callers.

8.2.2 Results - Deletions

The evaluated deletion caller is based on the first algorithm described in section 4.4.1, the specific parameters are the following :

- At least 4 overlapping reads with an inferred fragment size that is unexpectedly large must be present in a given region to start analysis. This uses the signal that counts the number of reads with an inferred fragment size too big as an input and filters based on a threshold of at least 4 reads.
- The mates of the reads must be clustered together by no more than 3 average fragment lengths. This is a second filter applied to the regions remaining after the first filter.
- The inferred size of the fragments must not be more than 1'000'000 bases (arbitrary filter).
- Based on the coverage between the reads and their mates the event is classified as a deletion (both alleles) if the coverage is below half of the average coverage, deletion (single allele) if the coverage is below 2/3 the average coverage, or classified as an uncalled structural variant if the coverage is higher (could be translocation or other event).
- The calls are then filtered to keep only the ones called as deletions.

This very simple deletion caller was implemented and benchmarked on the simulation dataset and the results are compared to the state of the art (best deletion callers) in table 8.1.

SV Type	Tools	Precision	Recall
DEL	This work	80.4%	76.6%
	GRIDSS	98.9%	86.6%
	Lumpy	99.1%	81.4%
	SVSeq2	96.2%	86.1%
	SoftSV	96.8%	83.6%
	Manta	95.9%	83.1%
	MATCHCLIP	99.4%	71.7%
	inGAP-sv	91.1%	78.6%

Table 8.1 Precision and Recall of our simple deletion caller against the state of the art

Precision, number of correct calls over all calls made, true positives/(true positives+false positives) and Recall, number of correct calls over all deletion events, true positives/(true positives+false negatives) are not as high as the best state of art software but not that far behind.

Explorations of the results

Table 8.2 shows the detailed results of this simple deletion caller on a per chromosome basis. The performance is roughly similar on all chromosomes, except Y. Mainly because of the low number of events, resulting in a low precision score due to the relatively high number of incorrectly predicted calls. In order to get a better understanding as to why calls were missed on incorrectly predicted we will explore the results of chromosome 21, chosen arbitrarily because of the relatively low number of events in the truth set (57).

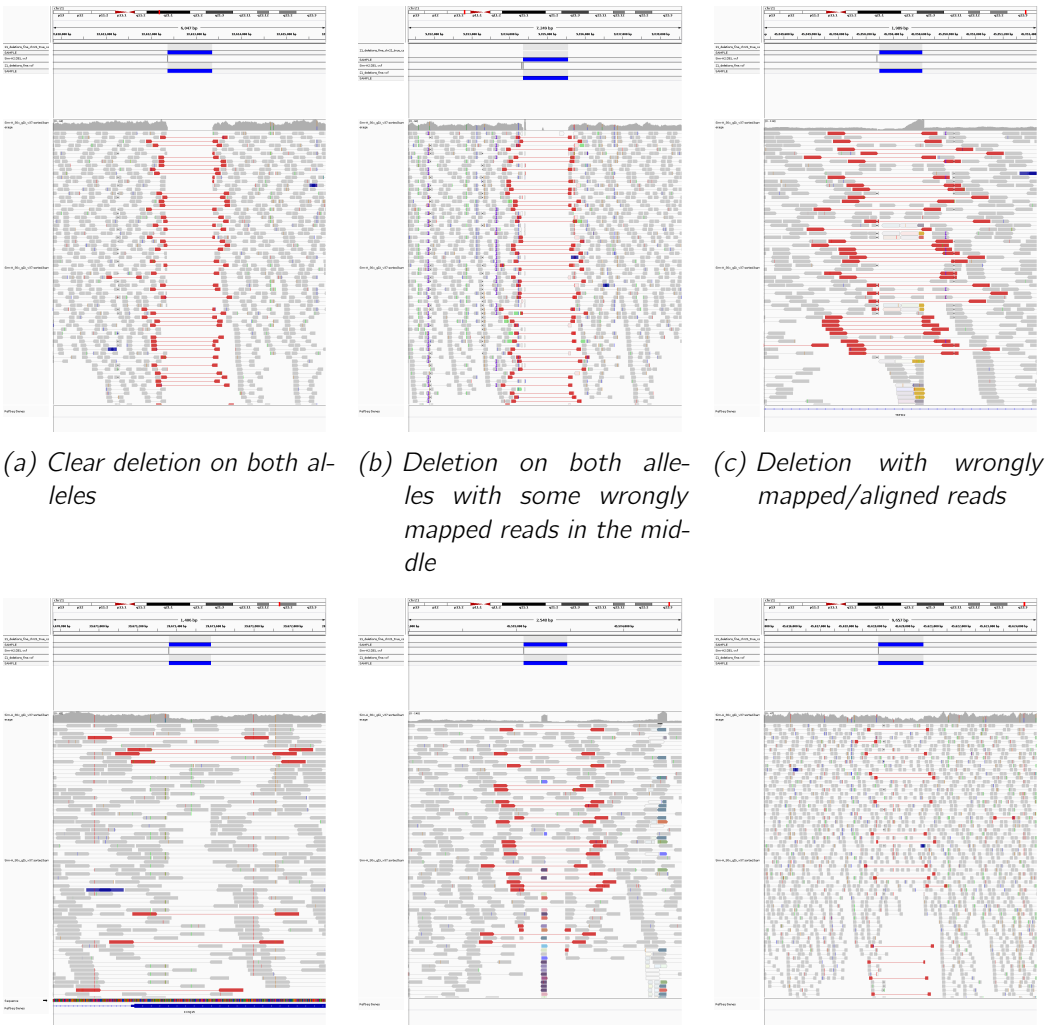
Chr	Number of events in truth set	Correctly Predicted (TP)	Missed (FN)	Incorrectly Predicted (FP)	Precision TP / (TP + FP)	Recall TP / (TP + FN)
All	**3530**	**2704**	**826**	**659**	**80.4%**	**76.6%**
1	232	180	52	59	75.3%	77.6%
2	251	194	57	66	74.6%	77.3%
3	196	147	49	49	75.0%	75.0%
4	258	203	55	38	84.2%	78.7%
5	192	151	41	43	77.8%	78.6%
6	219	177	42	38	82.3%	80.8%
7	210	159	51	33	82.8%	75.7%
8	202	157	45	26	85.8%	77.7%
9	128	89	39	28	76.1%	69.5%
10	186	149	37	23	86.6%	80.1%
11	165	132	33	17	88.6%	80.0%
12	176	133	43	30	81.6%	75.6%
13	122	87	35	20	81.3%	71.3%
14	127	102	25	16	86.4%	80.3%
15	87	70	17	6	92.1%	80.5%
16	115	80	35	29	73.4%	69.6%
17	104	80	24	20	80.0%	76.9%
18	100	80	20	21	79.2%	80.0%
19	133	93	40	24	79.5%	69.9%
20	72	48	24	16	75.0%	66.7%
21	57	46	11	8	85.2%	80.7%
22	95	65	30	18	78.3%	68.4%
X	100	80	20	25	76.2%	80.0%
Y	3	2	1	6	25.0%	66.7%

Table 8.2 Per chromosome decomposition of the results of the deletion caller

Examples of correct calls

The VCFotographer tool was run on the VCF files for chromosome 21, first some correctly predicted events will be shown in order to give an idea of how the events look and in what they differ with the missed calls or wrongly predicted calls.

Figures 8.1a to 8.1f show examples of deletions that were correctly predicted on chromosome 21 of the simulation data set.



(a) Clear deletion on both alleles

(b) Deletion on both alleles with some wrongly mapped reads in the middle

(c) Deletion with wrongly mapped/aligned reads

(d) Clear deletion on a single allele

(e) Deletion on a single allele with incorrectly mapped reads in the middle

(f) Deletion on a single allele with little effect on read depth

Figure 8.1 Examples of correctly called deletions on the simulation data set chromosome 21

The diversity of the deletions called correctly shows the versatility of the detection method implemented by this caller.

Exploration of missed calls

There are 11 missed calls (false negatives) on chromosome 21, they are explored one by one below and it explained as to why they were missed. The first thing to do was to filter missed calls and calls that had an overlap that did not satisfy the minimum reciprocal overlap conditions. From here we already find that 5 deletions are found but have imprecise breakpoints. Figure 8.2a to 8.2e show these cases. This suggest that improvements could be made on how the breakpoints are computed.

(a) Deletion example with imprecise breakpoints

(b) Deletion example with imprecise breakpoints

(c) Deletion example with imprecise breakpoints

(d) Deletion example with imprecise breakpoints

(e) Deletion example with imprecise breakpoints

Figure 8.2 Examples of false negative deletion calls that did not satisfy the reciprocal overlap criterion

This leaves us with 6 missed calls that we can now explore in more detail.

1. Figure 8.3 shows a small single allele deletion of 80 bases. This was completely missed by the algorithm because the size is smaller than the variability of the inferred fragment sizes, also because of the location of the deletion the impact on the coverage is very small. Therefore this deletion case would also be difficult to find with read depth based algorithms.

Figure 8.3 Small 80 base deletion on a single allele, very difficult deletion to find for any algorithm

2. Figure 8.4 shows a deletion that was predicted by the algorithm but filtered out when doing the calling because the coverage between the breakpoints was too high, the caller did generate a call but marked it as "SV" and not "DEL" because it could not infer that it was a deletion. These extra calls are kept in a separate VCF file, they contain predictions based on the insert size that were not clear deletions (can also contain for example intrachromosomal translocations or more complex events).

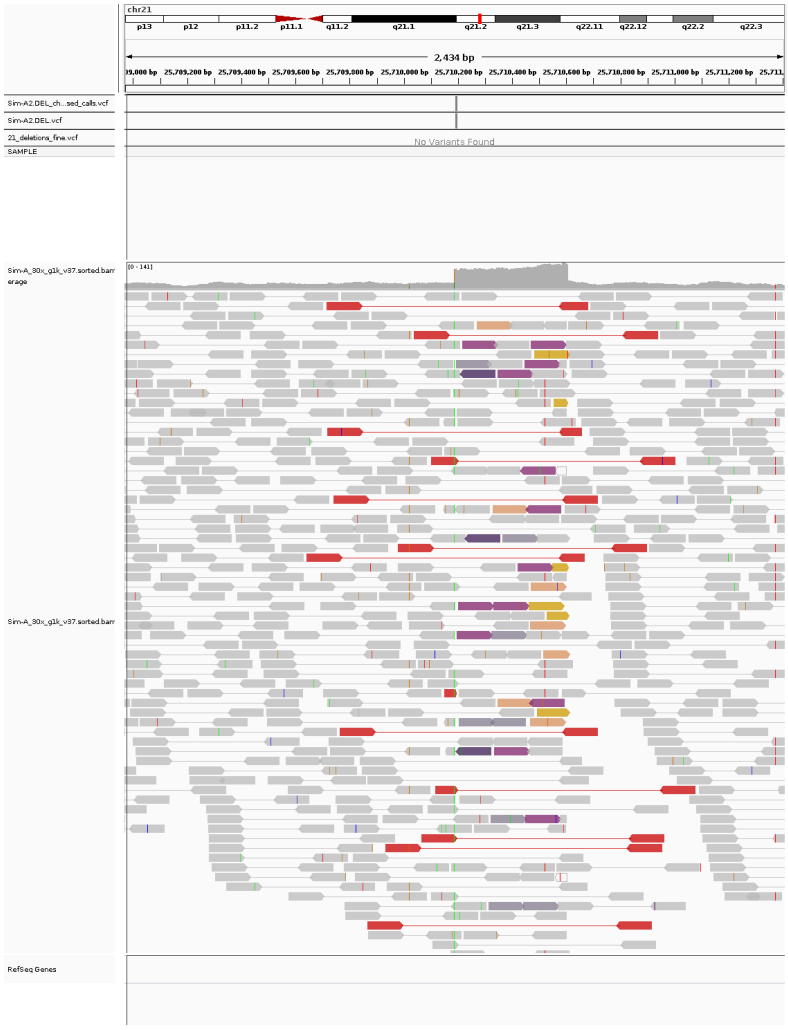


Figure 8.4 Unexpectedly high coverage in deletion site

3. Figure 8.5 shows a missed deletion that seems like a clear single allele deletion. This region was predicted in the algorithm but during analysis the breakpoints could not be clearly extracted because some reads have multiple mappings (secondary alignments) on both sides of the deletion (split reads) therefore the simple algorithm that predicts the breakpoints from the rightmost forward reads and leftmost reverse reads could not generate a correct interval. This region was called as "SV" because the interval (start and end breakpoints) could not be predicted. This suggest again that improving the part of the algorithm that extracts the breakpoints would improve this caller.

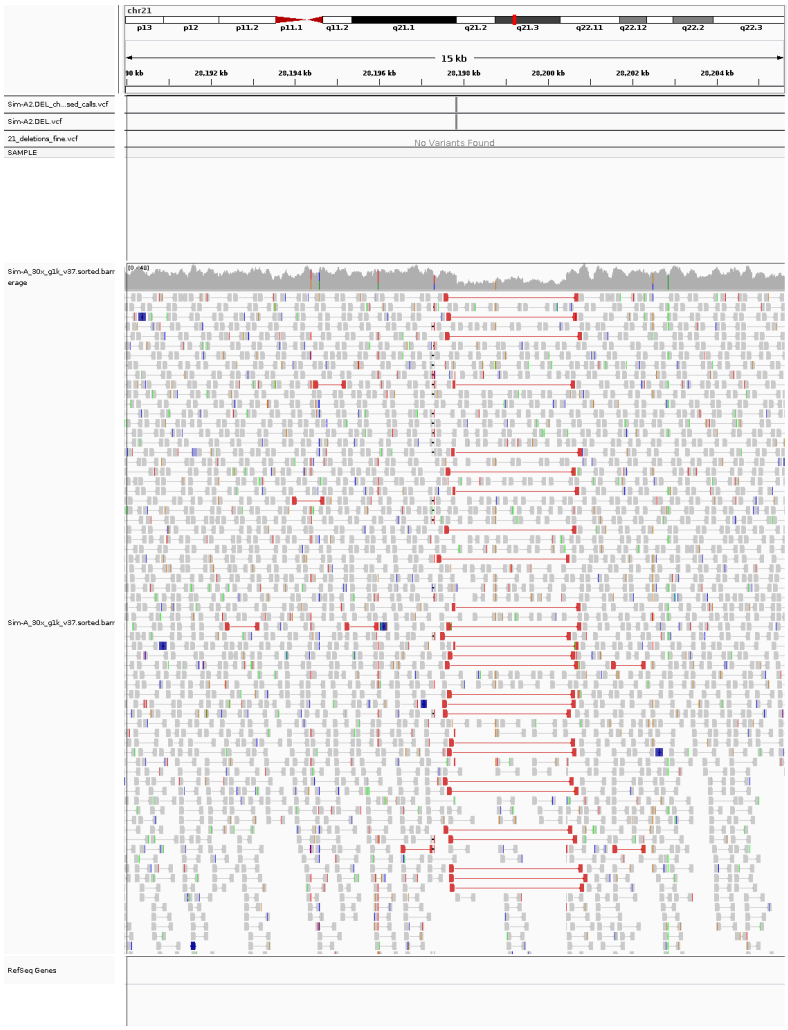


Figure 8.5 Seemingly clear single allele deletion

4. Figure 8.6 shows a 104 base deletion that was missed by the caller. The deletion is a small deletion of 104 bases, which means that it did not affect the inferred fragment sizes enough to provide clear evidence of a deletion. The region also has an unexpected coverage and some aligned reads show insertions and deletions in their alignments. This makes this event a perfect candidate for de-novo assembly because of the multiple signs of variation but unclear result.

Figure 8.6 Hard to find deletion of 104 bases

5. Figure 8.7 shows a region where a 63 base deletion was simulated on a single allele. Except from the unexpected coverage there are no clearly visible signs of this deletion. Again this type of event would require de-novo assembly to solve. The assembly should also be able to handle differentiating two possible assemblies, the paternal and the maternal alleles. Based on the unexpected coverage this regions can be predicted as a region with signs of structural variation, however to solve the case more involved methods are required.

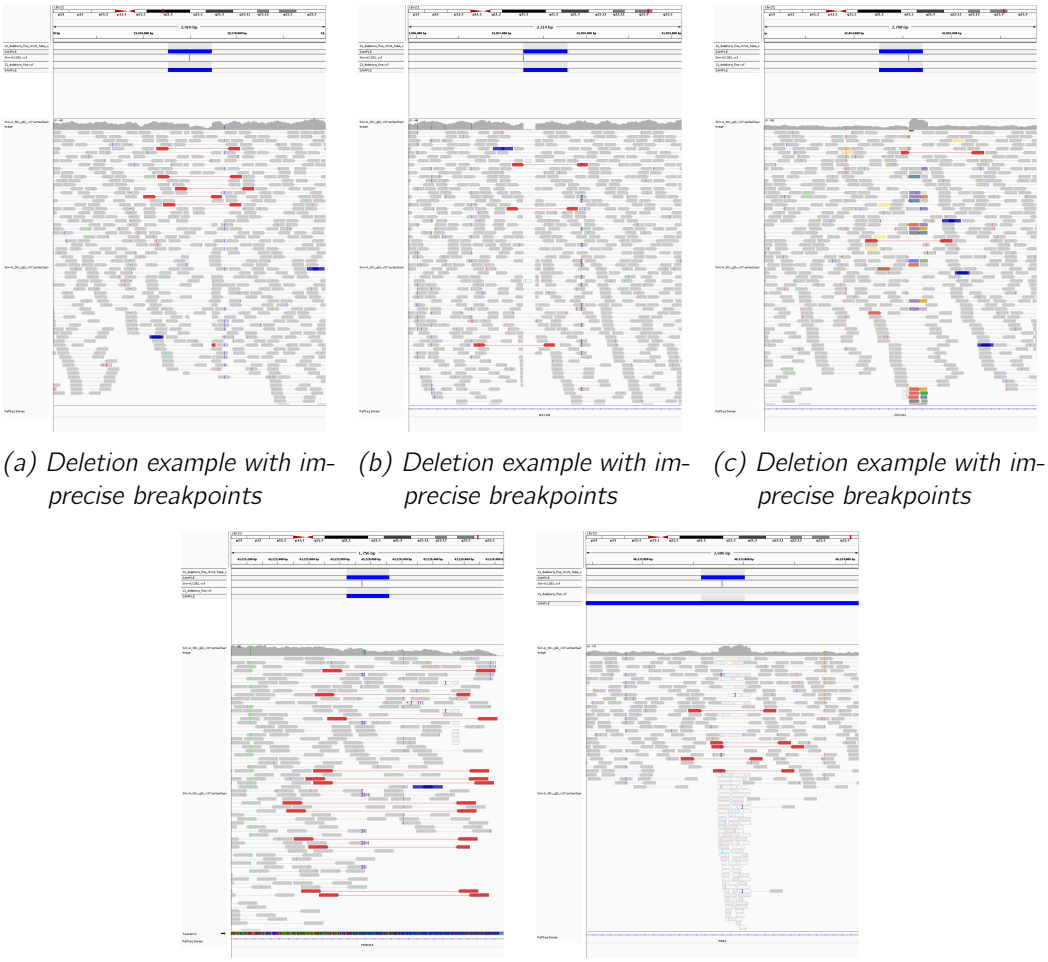
Figure 8.7 Hard to find deletion of 63 bases

6. Figure 8.8 shows a 109 base deletion on both alleles. There was not enough evidence (number of pairs with inferred fragment size that is too big) for the caller to detect the event. Lowering the threshold of number of pairs required to begin analyzing the region for deletions would allow to find this event, improving detection recall but would also likely reduce precision because of possible extra false positives.

Figure 8.8 Deletion of 109 bases on both alleles that was missed by the deletion caller

Exploration of false positive calls

Wrong calls or false positive calls are when the caller predicted a deletion but in fact there was no deletion event at the location. Of the 8 calls that were classified as false positives 5 were in fact correctly predicted but had imprecise breakpoints and did not satisfy the reciprocal overlap criterion. Figures 8.9a to 8.9e show the cases that did not satisfy the reciprocal overlap criterion improving the calling of the breakpoint locations of the event would solve these cases.



(a) Deletion example with imprecise breakpoints

(b) Deletion example with imprecise breakpoints

(c) Deletion example with imprecise breakpoints

(d) Deletion example with imprecise breakpoints

(e) Deletion example with imprecise breakpoints

Figure 8.9 Examples of deletion false positive calls that did not satisfy the reciprocal overlap criterion

Finally only three cases of false positives remain. Figures 8.10 and 8.11 show two of the three remaining cases that were wrongly reported as deletions. These false positive calls arose from edge cases with inferred fragment size. The expected fragment size thresholds were computed from the distribution of the 10'000 first reads in the alignment file for simplicity instead of e.g., locally on the current chromosome. This is also why they do not appear in red in IGV (IGV does a local computation). Therefore, these cases could be solved by computing the thresholds for the fragment sizes better.

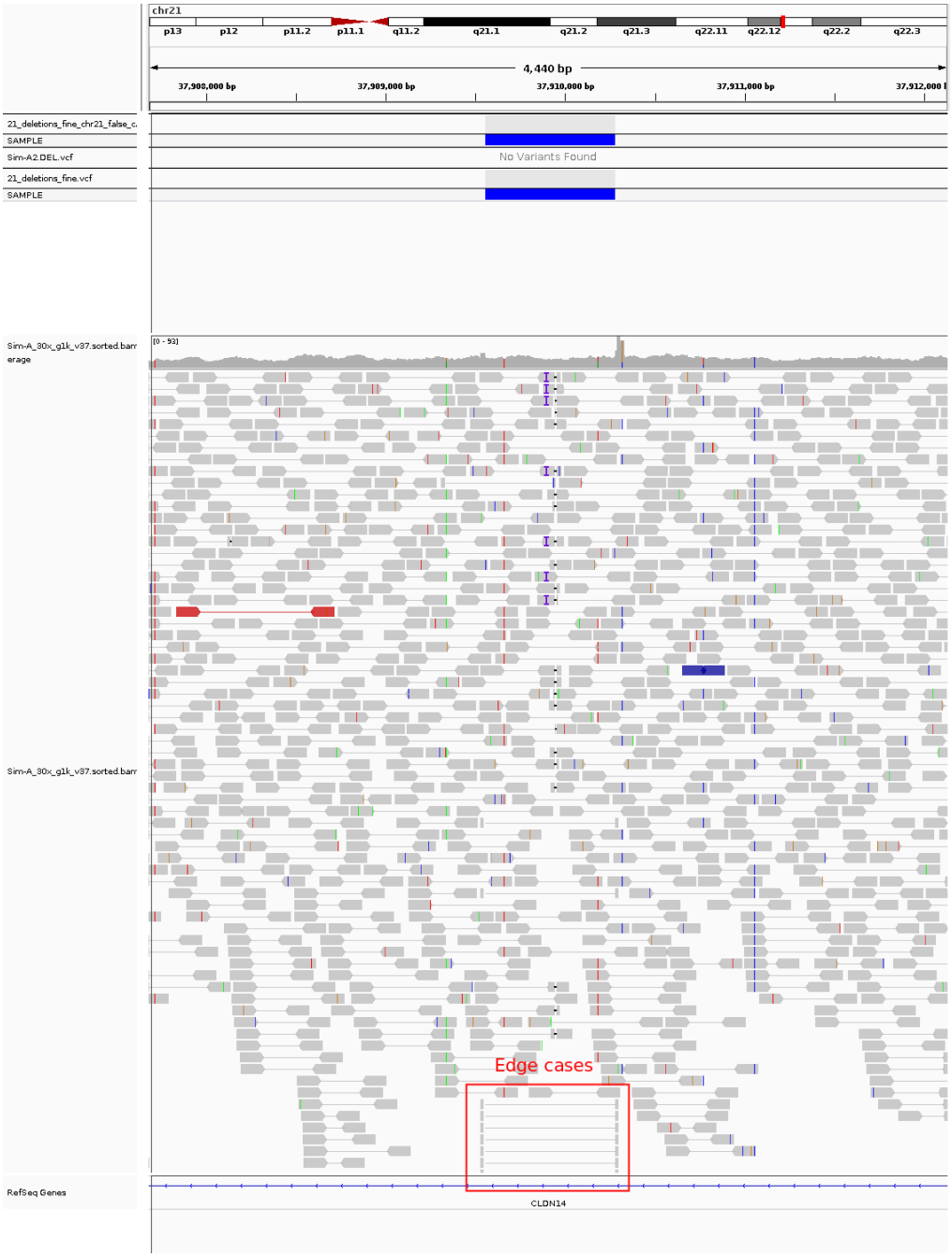


Figure 8.10 False positive deletion call due to edge cases of inferred fragment length

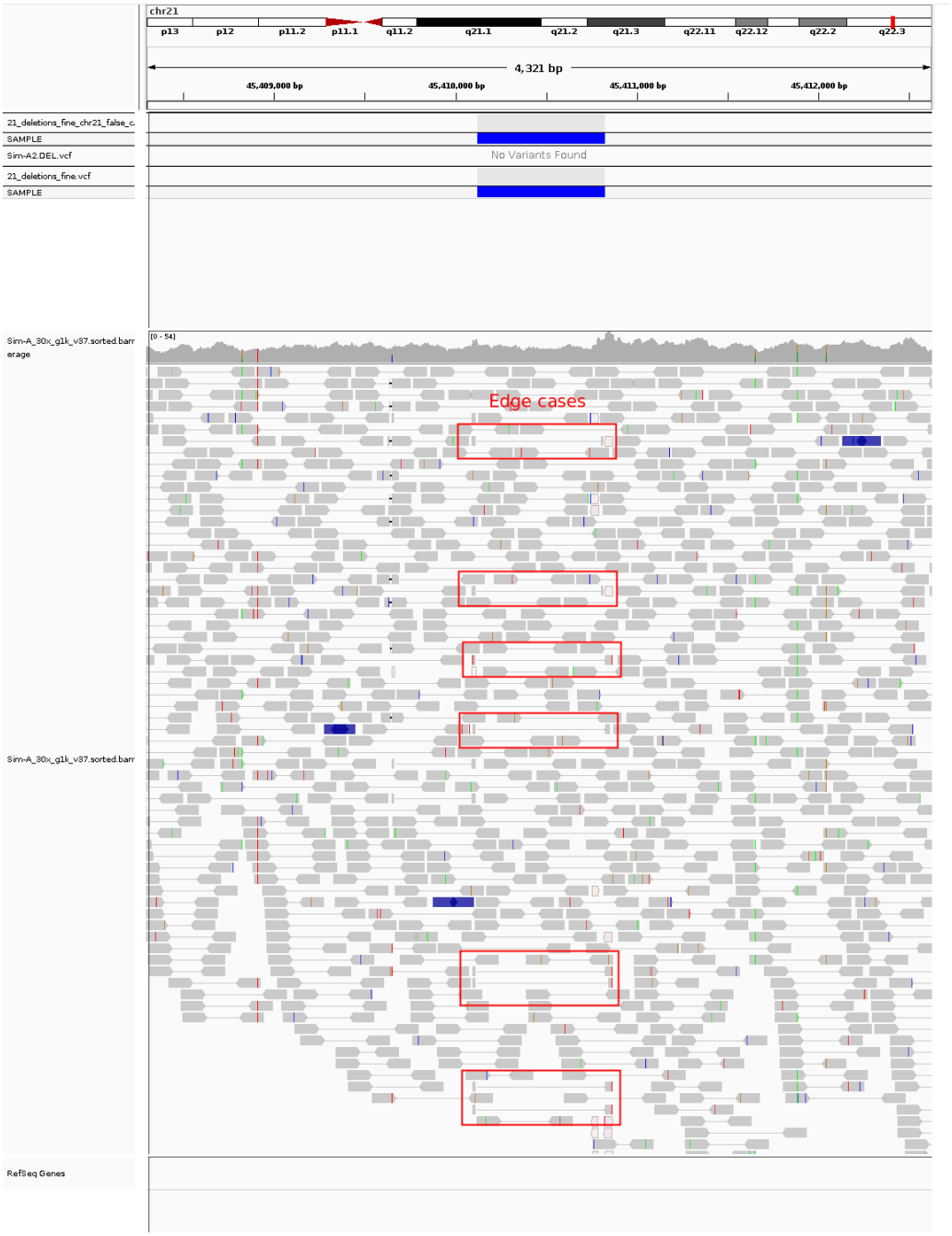


Figure 8.11 False positive deletion call due to edge cases of inferred fragment length

The last false positive event shown in figure 8.12 is due to incorrectly mapped reads (not shown in this figure because of scale). The reads that span this big region, over 51 kilo bases in size, are part of fragments that overlap this region. Other big deletions of tens of thousands of bases were detected correctly in this data set, however this one is not. Figure 8.13 shows the fragments at one end (visible by the long red lines). One thing that is a warning sign is the high coverage at the breakpoint on the right. Figure 8.14 shows both breakpoints in a split view. The high number of reads (high coverage) and the high number of errors on the aligned reads (colored vertical stripes) are signs that these reads are probably incorrectly mapped. This can happen when reads that overlap a variation breakpoint don't share enough similarity to the reference and get mapped at incorrect places, leading to wrong calls such as this one. Filtering based on this observation could allow to remove false positives like this one.

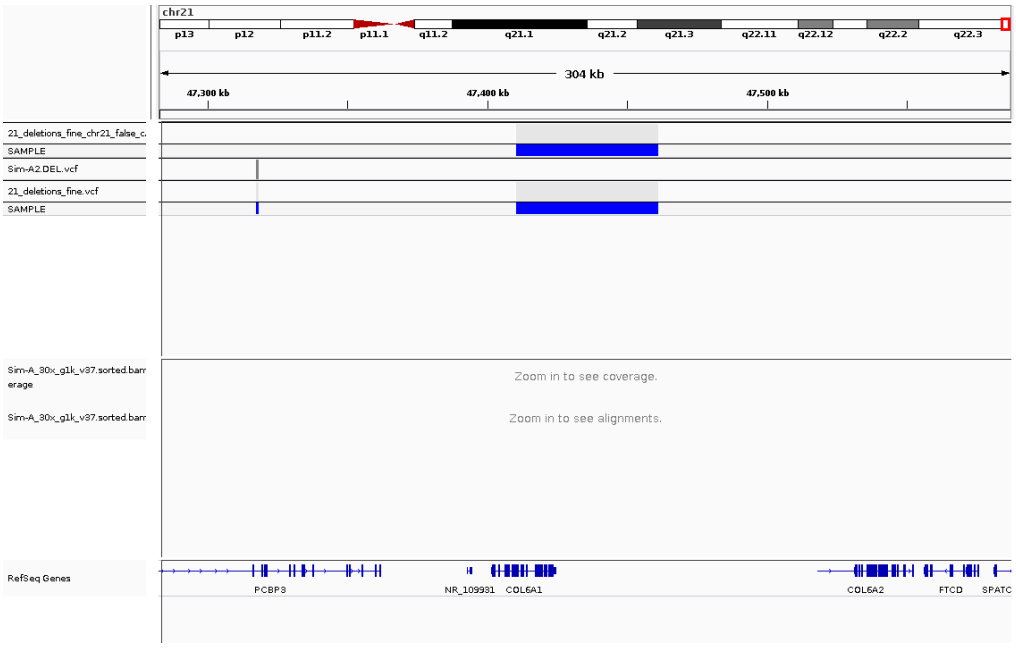


Figure 8.12 Large false positive deletion call

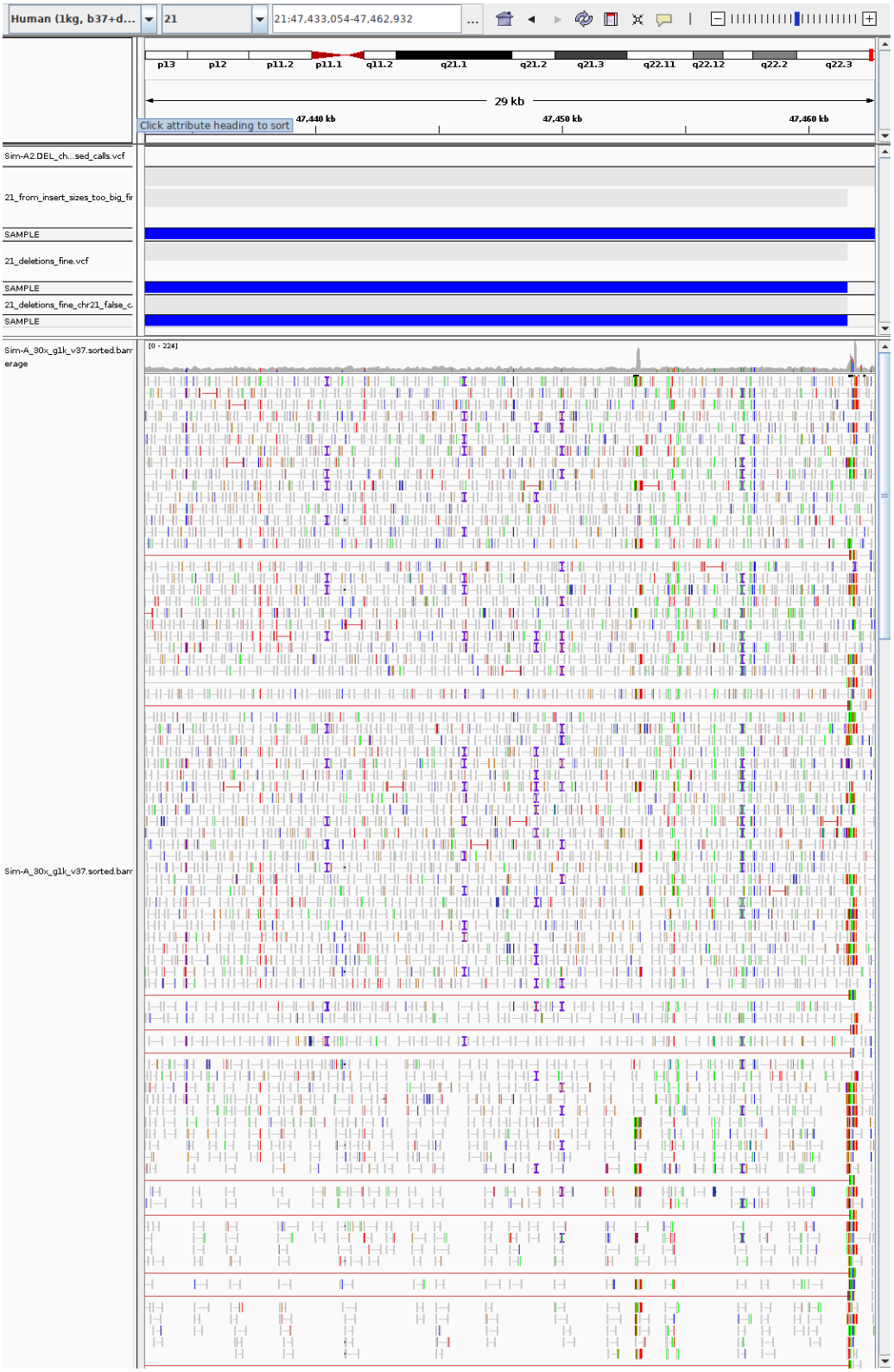


Figure 8.13 Large false positive deletion call rightmost breakpoint

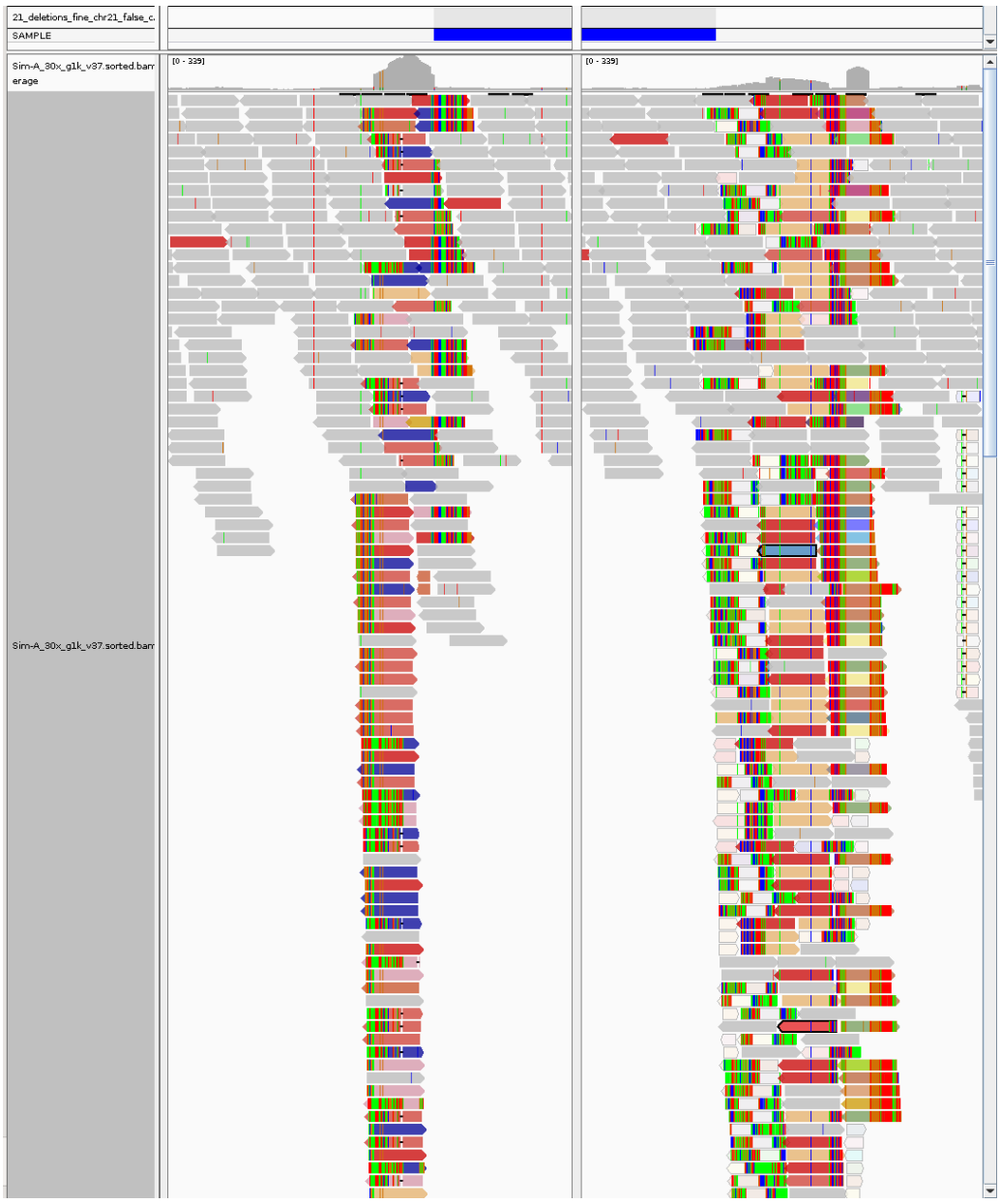


Figure 8.14 *Large false positive deletion call both breakpoints shown*

Exploration of results conclusion

This gives us a good idea of the different possible strategies to improve the deletion calling algorithm. Refining the breakpoint computation to be more precise will improve the overall performance the most. A majority of the wrong calls (false negatives and false positives) were in fact simply calls that did not satisfy the reciprocal overlap and could not be considered to be the same call as the expected call of the truth set. Other cases are more complex and would require filtering based on a more in-depth analysis or would require to look at other signals for detection, e.g., deletions in the alignments (CIGAR strings). Some events could also benefit from de-novo assembly to clearly find out what is happening.

Overall for a simple deletion caller based on the inferred fragment length without any tuning and optimization the results are good and not far behind the state of the art.

8.2.3 Results - Duplications

A duplication caller was created by applying the mathematical morphology algorithm described in section 4.4.3 on the coverage signal. The coverage signal was segmented based on edges and region with a coverage of 0. The resulting segments were averaged to generate an approximate coverage signal that correlates with the copy number.

The resulting signal was filtered so that any region with coverage above the expected coverage (above 1.5 times the mean coverage of the chromosome) was marked as high coverage. The marked regions were joined if the regions were apart by less two reads of distance to be finally called as duplications.

This duplication caller is simplistic and unrefined. Also it does not apply any hierarchical analysis in order to separate single allele duplications. This means it will be wrong every time heterozygous duplications overlap and will be wrong every time the segmented regions are split or joined incorrectly.

This caller was developed in very little time in order to show results for a possible duplication caller developed and benchmarked with the framework. This is why the algorithm is so simple and does not take into account any particularities that could be found around duplication sites.

Table 8.1 shows the results against the state of the art.

SV Type	Tools	Precision	Recall
DUP	This work	53.8%	68.7%
	Wham	96.9%	81.7%
	SoftSV	84.2%	67.8%
	MATCHCLIP	87.6%	77.5%
	GRIDSS	91.1%	77.9%
	Manta	99.0%	83.2%
	SvABA	82.6%	69.6%

Table 8.3 Precision and Recall of our simple duplication caller against the state of the art

This caller has a low precision meaning it has too many false positives. This is mostly because there are many regions were the coverage is high because of mismapped reads (as was visible in some deletion examples above). These cases could be filtered out but the algorithm returns every region where the coverage was higher than expected, resulting in the poor precision figure above.

Anyway, this crude duplication caller can serve as a basis to create better callers and shows that even an extremely simple algorithm developed in less than an hour with no specific analysis is already able to detect several duplication sites correctly.

Exploration of the results

Table 8.4 shows the detailed number of calls for each chromosome. This simple caller could correctly call about two thirds of the events in the truth set. However the precision is very low because many of the calls made are in fact incorrect.

Chr	Number of events in truth set	Correctly Predicted (TP)	Missed (FN)	Incorrectly Predicted (FP)	Precision TP / (TP + FP)	Recall TP / (TP + FN)
All	**1656**	**1137**	**519**	**978**	**53.8%**	**68.7%**
1	122	73	49	72	50.3%	59.8%
2	126	92	34	77	54.4%	73.0%
3	110	84	26	54	60.9%	76.4%
4	81	58	23	59	49.6%	71.6%
5	105	75	30	50	60.0%	71.4%
6	76	51	25	61	45.5%	67.1%
7	104	64	40	67	48.9%	61.5%
8	83	61	22	44	58.1%	73.5%
9	92	64	28	41	61.0%	69.6%
10	85	56	29	45	55.4%	65.9%
11	65	42	23	33	56.0%	64.6%
12	59	41	18	42	49.4%	69.5%
13	50	42	8	26	61.8%	84.0%
14	59	42	17	17	71.2%	71.2%
15	65	37	28	33	52.9%	56.9%
16	67	42	25	36	53.8%	62.7%
17	78	54	24	38	58.7%	69.2%
18	39	27	12	25	51.9%	69.2%
19	50	34	16	36	48.6%	68.0%
20	30	20	10	18	52.6%	66.7%
21	17	15	2	14	51.7%	88.2%
22	34	21	13	38	35.6%	61.8%
X	54	41	13	43	48.8%	75.9%
Y	5	1	4	9	10.0%	20.0%

Table 8.4 Per chromosome decomposition of the results of the duplication caller

Chromosome 21 was chosen to show some examples of the results because of the small number of events. The two missed calls were in fact called but did not satisfy the reciprocal overlap criterion. This is because the segmentation and join conditions of the coverage signal were chosen without much care. Figure 8.15 shows the first missed duplication call because the segmentation created an interval that was too short. The event in itself is found but the breakpoints are incorrect.

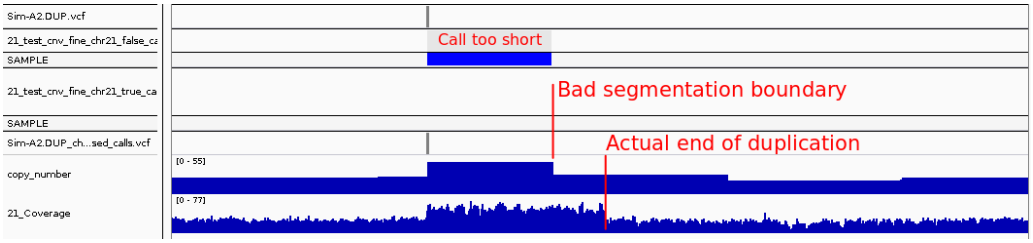


Figure 8.15 Missed duplication call because of incorrect breakpoints (call too short)

Figure 8.16 shows the second missed call, this time the call made was actually too long, this is another example of wrongly called breakpoints of a true event that was detected.

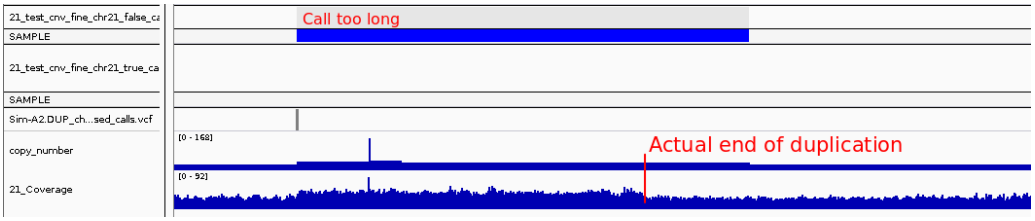
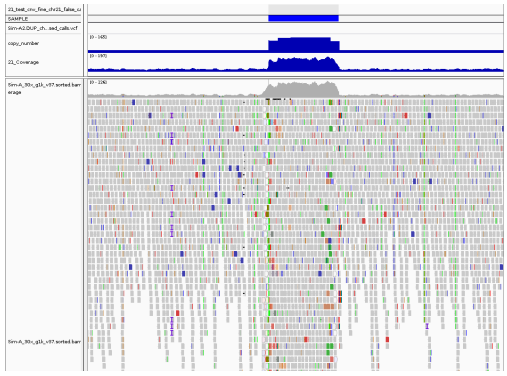


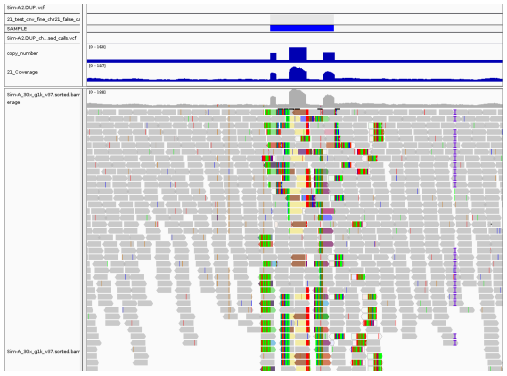
Figure 8.16 Missed duplication call because of incorrect breakpoints (call too long)

This shows that refinement of the segmentation algorithm or adding a second pass of an analysis algorithm around the calls made by this crude caller would allow to solve most cases, the same could apply to the false positives discussed below.

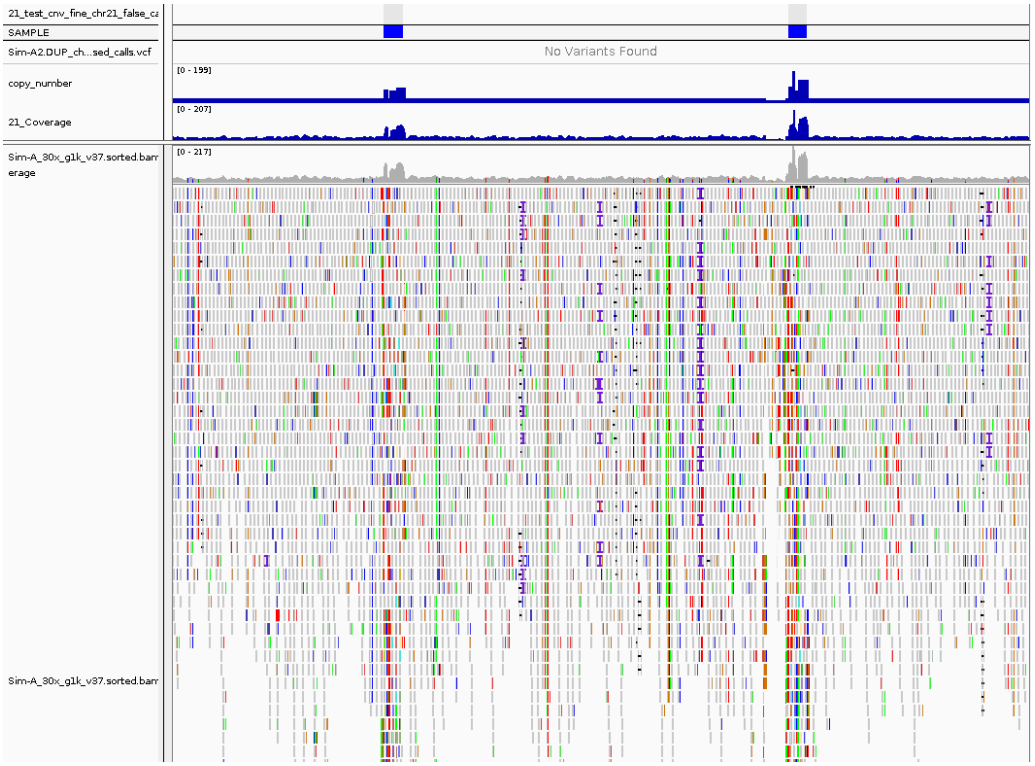
The high number of false positives is due to the fact that many regions have an unexpectedly high coverage. This high coverage is most likely due to reads mapped to incorrect locations. This can be seen in figures 8.17a to 8.17c that show 4 examples of the incorrectly predicted calls (false positives).



(a) Duplication false positive



(b) Duplication false positive



(c) Duplication false positives

Figure 8.17 Examples of duplication false positive calls

These examples give us a good idea how to reduce the number of false positives, from the examples above we can see that many of the mapped reads are either clipped (split reads) or have their mate on another chromosome (colored reads). Therefore one relatively simple approach could be to take all the calls from the crude duplication caller and for each call query the reads from the alignment to evaluate if they are split reads or have their mate mapped onto another chromosome. If the number of reads that show these kind of properties is above a threshold the call should be discarded. The regions predicted by this caller may still be of interest for other analysis but they are not duplications. Another approach to do this filtering would be by using a second

order signal that is the coverage minus the number of reads that are split (clipped) or have their mate on another chromosome. Both of these signals can be or are already generated by the signal extractor tool.

Figure 8.18 shows the same two last false positive events with the clipped read and the interchromosomal read-pair signals. Reads exhibiting these properties, probably mismapped, are the reason behind most of the false positives for this caller. Figure 8.19 shows a correct call with a spike in the coverage due to clipped and interchromosomal reads.

Figure 8.18 False positive duplication calls with clipped and interchromosomal signals

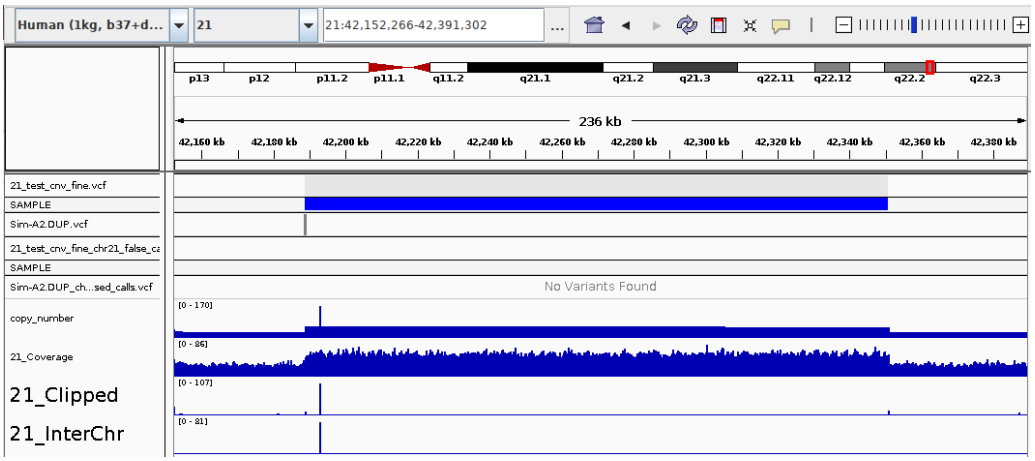


Figure 8.19 True positive duplication call with clipped and interchromosomal signals

These examples show the high potential of the framework for developing better callers. Even when the results of the caller are not great, the visualization and interpretation of signals shows how easy it would be to tune and refine the caller within this framework.

Especially given the fact that because the signals are already generated running the caller algorithms is done very quickly compared to running a caller that does the signal and feature extraction directly from the BAM file. More details on the time required to run the callers and signal extraction respectively are given later in this chapter.

Benchmarking a caller not only results in the statistics as shown in table 8.4 but also in the variant call files for all the correctly predicted calls (true positives), missed calls (false negatives) and incorrectly predicted (false positive) calls. These files can then be used with the framework for an interactive visualization session in IGV helping the developer to interpret the results. It is also possible to automatically generate a collection of captures with the VCFotographer tool to be interpreted later. The real strength becomes apparent when relevant signals are loaded besides the reads as shown above. This can make the reasons behind the performance of a given caller very clear, much clearer than looking at the statistics alone and having to tune parameters in the dark.

8.2.4 Results - Inversions

An inversion caller was created following the algorithm described in section 4.4.4. The specific parameters are the following :

- The minimum number of overlapping reads that have an incoherent pair orientation must be at least 4 for a region to be considered.
- The regions are filtered based on this predicate applied on the signal that counts the number of reads with incoherent pair (tandem) orientation. The resulting intervals are joined if they are less than a distance of two read lengths apart.
- All these regions are linked together by the read pairs that span two regions and marked as inversions. (For all regions, extract reads that are in a discordant orientation pair and create links between the regions based on those pairs).
- Filter regions so that only couples of regions linked by at least 10 read pairs remain.
- Inversions breakpoints are inferred from the center of the distribution of the reads on each side of the inversion regions (linked pairs of regions).

Table 8.5 shows the results of the inversion caller described above against the state of the art. The results are close and sometimes even better to the best existing callers.

SV Type	Tools	Precision	Recall
INV	This work	94.0%	75.7%
	DELLY	94.7%	81.8%
	TIDDIT	89.2%	77.9%
	1-2-3-SV	70.7%	81.2%
	GRIDSS	96.6%	84.7%

Table 8.5 Precision and Recall of our simple inversion caller against the state of the art

The results of this inversion caller are very good but can still be improved, the main weakness of this caller is a slightly low recall meaning it missed several events. Below we will explore why.

Exploration of the results

Table 8.6 shows the results of the inversion caller on a per chromosome basis.

Chr	Number of events in truth set	Correctly Predicted (TP)	Missed (FN)	Incorrectly Predicted (FP)	Precision TP / (TP + FP)	Recall TP / (TP + FN)
All	**309**	**234**	**75**	**15**	**94.0%**	**75.7%**
1	29	22	7	3	88.0%	75.9%
2	22	17	5	1	94.4%	77.3%
3	17	14	3	0	100.0%	82.4%
4	10	7	3	0	100.0%	70.0%
5	12	6	6	0	100.0%	50.0%
6	14	11	3	1	91.7%	78.6%
7	24	19	5	1	95.0%	79.2%
8	17	14	3	2	87.5%	82.4%
9	14	11	3	0	100.0%	78.6%
10	19	15	4	2	88.2%	78.9%
11	17	10	7	0	100.0%	58.8%
12	8	6	2	2	75.0%	75.0%
13	7	6	1	0	100.0%	85.7%
14	10	8	2	0	100.0%	80.0%
15	2	2	0	0	100.0%	100.0%
16	21	15	6	2	88.2%	71.4%
17	11	10	1	1	90.9%	90.9%
18	7	6	1	0	100.0%	85.7%
19	9	7	2	0	100.0%	77.8%
20	4	2	2	0	100.0%	50.0%
21	4	4	0	0	100.0%	100.0%
22	3	1	2	0	100.0%	33.3%
X	27	20	7	0	100.0%	74.1%
Y	1	1	0	0	100.0%	100.0%

Table 8.6 Per chromosome decomposition of the results of the inversion caller

Chromosome 8 was chosen arbitrarily for the exploration because it has 3 missed calls and 2 false positive calls. Figure 8.20 shows the first missed call, this inversion was actually called twice by the algorithm, however both calls did not satisfy the reciprocal overlap criterion and therefore were classified as false positives. These actually are the two false positives on this chromosome.

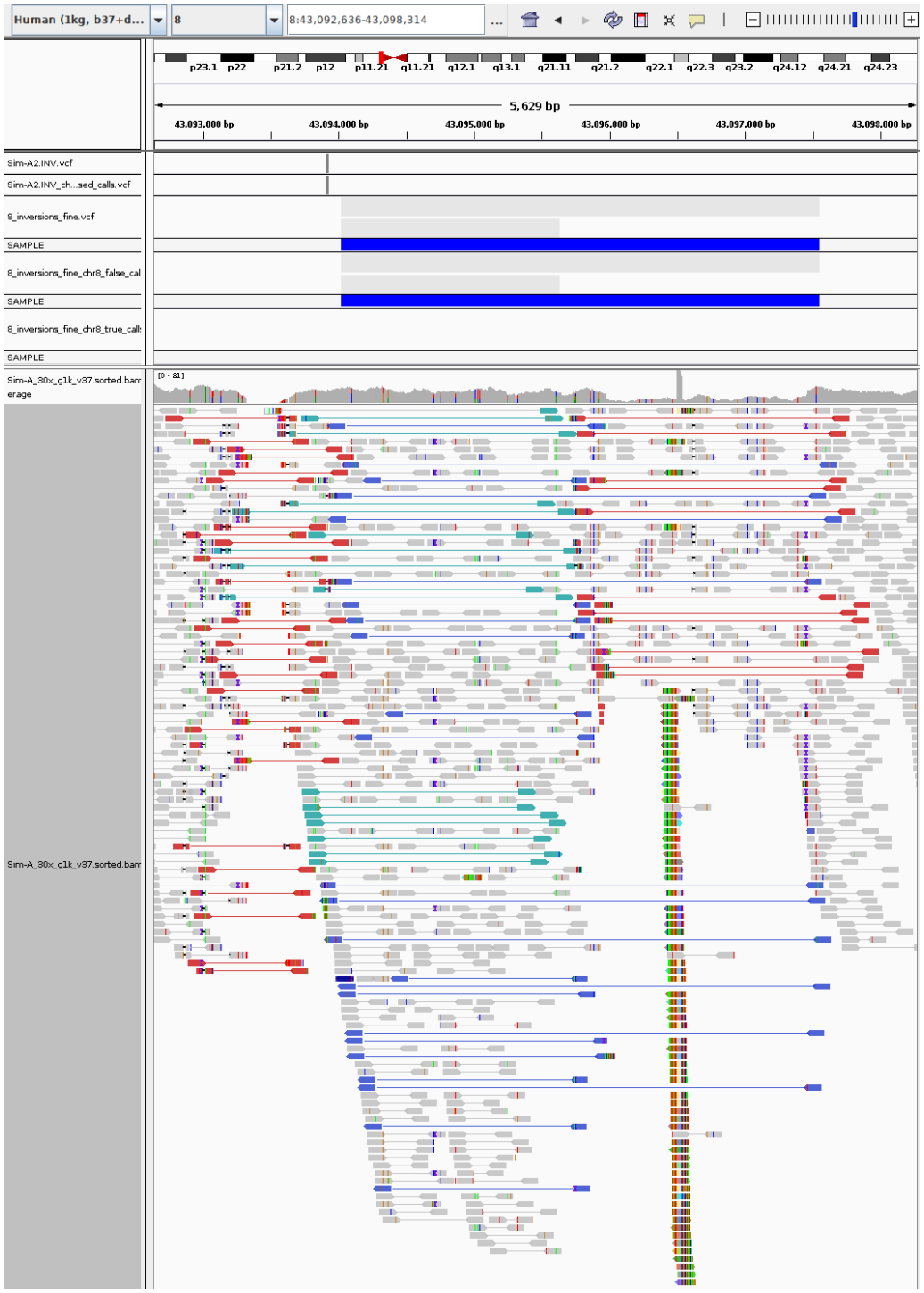


Figure 8.20 *Missed inversion call because it did not satisfy the reciprocal overlap criterion*

By looking at the results for the other chromosomes we can see that this prediction algorithm has a very high precision, meaning there are very few false positives. Further exploration of the other calls that were marked as false positives showed that they were also marked as false positives because they did not satisfy the reciprocal overlap

criterion. These calls do overlap true events but not well enough. This is also why the number of incorrectly predicted calls (false positives) is never higher than the number of missed calls (false negatives).

Figure 8.21 shows the second of the three missed inversions. This event was missed because the reads that span the breakpoints are mapped with a mapping quality score of 0 (reads colored very lightly) meaning that they map to several other places in the genome. Because the algorithm only considers reads with a non 0 mapping quality this event could not be found.

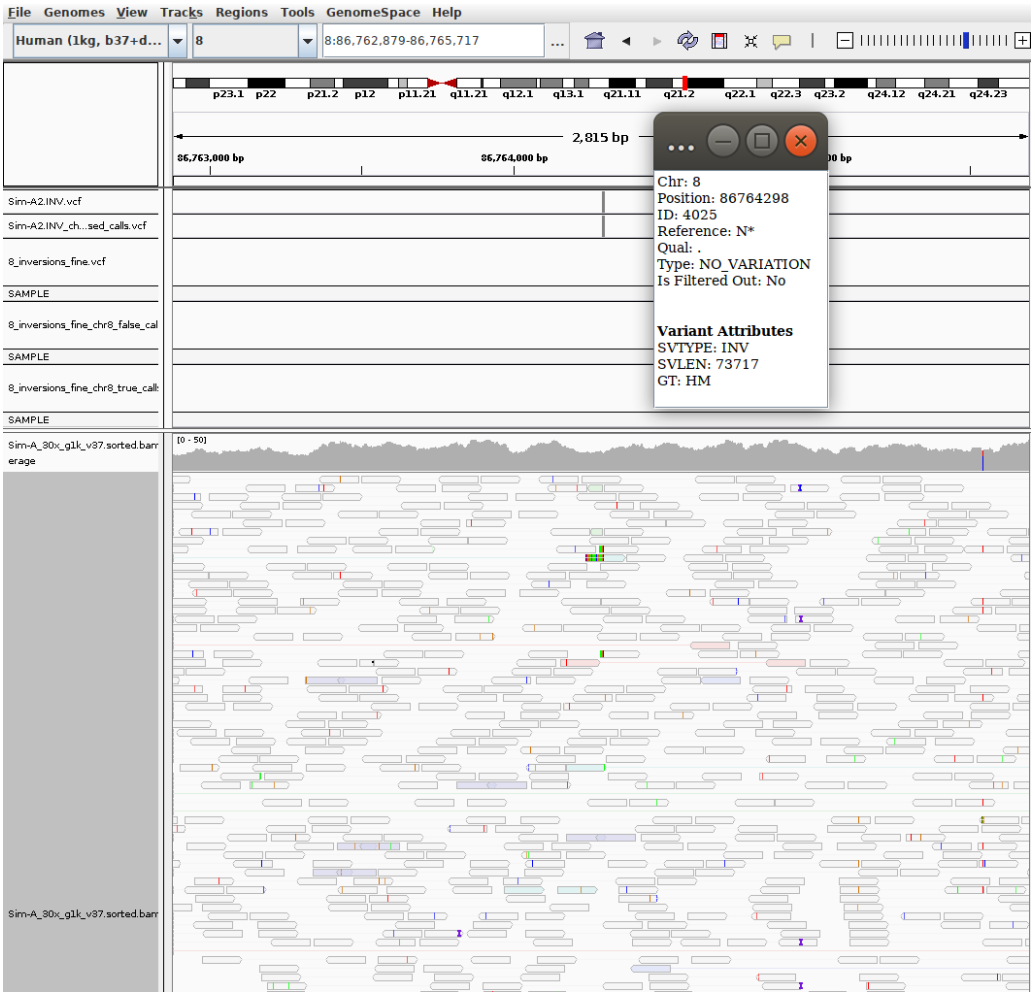


Figure 8.21 Missed inversion call because reads are mapped with mapping quality of 0

Figure 8.22 Shows the last missed inversion call on chromosome 8. This event was missed because it is a small inversion (247 bases) where the reads are clustered together. This resulted in the signal forming a single region that contained both ends of the pairs. This means that the region was linked to itself by the algorithm and discarded because it was linked to itself. These kind of events could be found by modifying the algorithm to keep pairs of linked regions even if the two regions of the pair are the same. The signal can be seen in figure 8.22. Another way to fix this problem would be

to remove the step that joins the filtered signal regions together if they are less than two read lengths apart (which is the case here). However, this may affect the other results negatively and should be benchmarked.

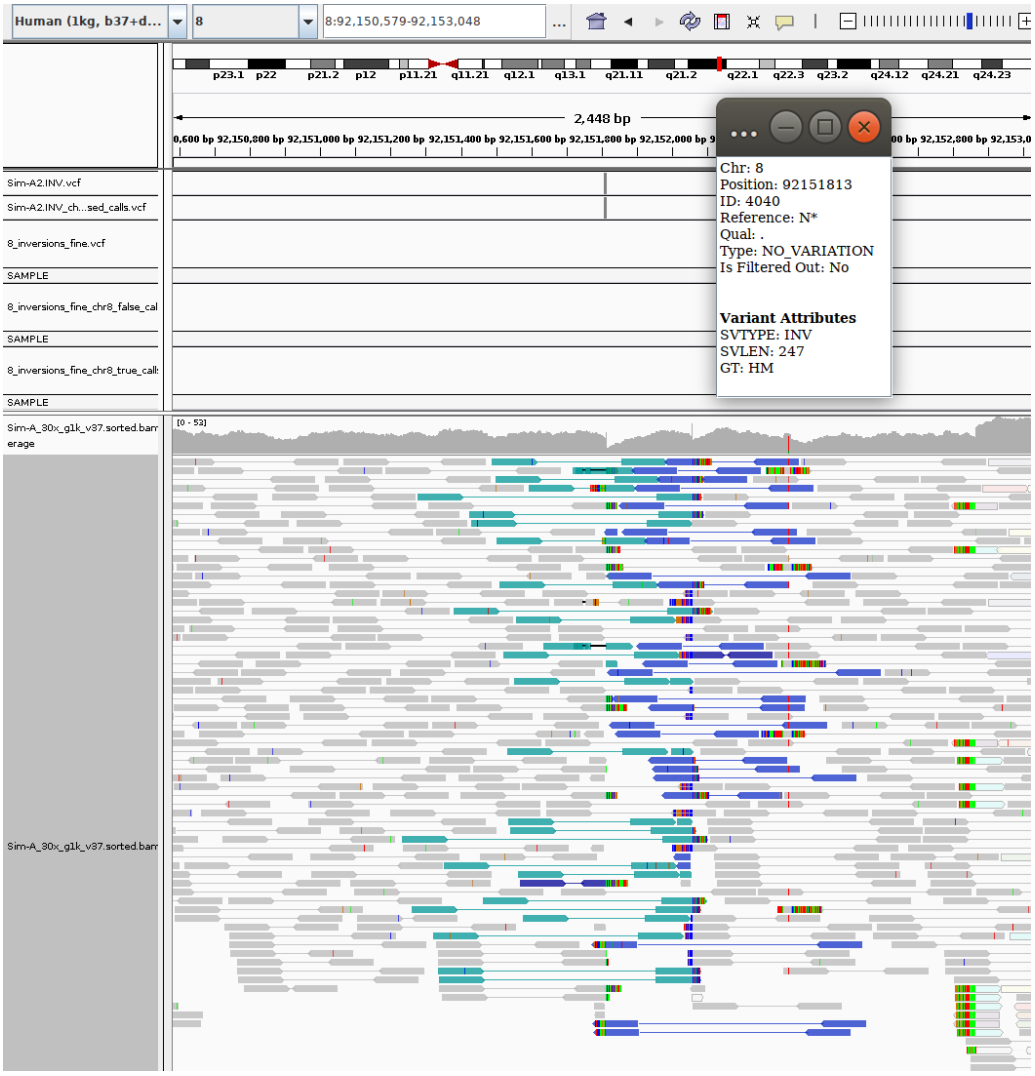


Figure 8.22 *Missed inversion call because whole region is considered as one end of the inversion*

Figure 8.23 *Missed inversion call with discordant pair orientation signal*

The nice thing about these incorrectly called results is that they show us very quickly what is wrong with the algorithm or why it was not able to generate the expected calls from the sequencing reads.

8.2.5 Results - Insertions

Insertions were not called because an insertion caller could not be completed during the time frame of this project, however several parts of the insertion caller exist, there are different predictors that can give regions that are susceptible to insertions based on the split (clipped) reads signal or the edit distance to the reference or edges in the coverage signal.

Also the definition of interval around an insertion is not clearly defined, the insertion often takes place at a specific place in the genome, which may be well defined for some cases but impossible to define for others (depending on the alignment the insertion may be placed somewhere else without being able to decide the exact location). The reciprocal overlap function is not necessarily well suited for the benchmarking of insertions. E.g., how should the size of the insertion impact this function. Also in order to compare insertions the content of the insertion should be analyzed and this is the part that is missing from the insertion caller in development. The local assembly algorithm was developed with the idea to solve the insertions from predicted regions, however while there is a predictor that can predict regions susceptible of insertions, which is presented in section 8.3.1 and a local assembly algorithm that can solve some regions, the complete caller was itself unfinished. Examples of assembled regions were shown in chapter 7.

These are the reasons why there are no results for insertion calling against the state of the art. The insertion calling is left for future works.

8.2.6 Summary of simulated dataset results

Three finished variation callers were implemented during the time frame of this project and tested on the simulation dataset. The deletion and inversion callers did perform well compared to the state of the art. The duplication caller did perform poorly. However exploration of the results gave us a lot of insight as to why it performed poorly. Analysis through visualization of the results showed us that given the framework and the already generated signals it would be possible to improve the callers easily. It would also allow to directly compare the resulting calls and statistics to the truth set and previous results. This gives us a perfect environment to develop and improve variant callers. Insertions could not be evaluated mostly because the time did not permit to develop a mature enough insertion caller. Nevertheless, insertion breakpoints can be predicted as will be shown in section 8.3.1. The majority of the extra work and analysis required would be to fully solve them and provide the sequence of the inserted sequence which is not always possible. The results show that it was possible to quickly develop variant callers for deletions, duplications and inversions given the framework that provided good results or at least insights as how to achieve good results.

8.3 Real Datasets

The real datasets used were from Genome In A Bottle (GIAB) samples. These have been sequenced by a multitude of technologies and GIAB developed benchmarks datasets for SV calls [191].

8.3.1 HG001 - NA12878

NA12878 has been characterized as the pilot genome by Genome In a Bottle and is one of the most, if not the most, sequenced sample. Samples from NA12878 can be obtained from several sources and sequencing data is available from GIAB for a plethora of sequencing technologies as well as data on repositories from laboratories and companies.

The samples come from a Utah (USA) woman. NA12878 has also been sequenced twice locally at the Genome Center using Illumina HiSeq2500 at 30x coverage. These datasets were used in the preliminary phase of this project during exploration and experimentation with signal extraction and variation prediction.

Insertions

Insertion regions were predicted using the generic algorithm presented in section 4.3.2. The algorithm did extract edges from the coverage signal starting from high increases in coverage and bridged them by regions that contained split (clipped) reads to end with high decreases in coverage.

Long read sequencing data available from GIAB was used in order to assess the relevance of the predictor visually, however no further benchmarking was realized because of time constraints. From the preliminary results it seems that the predictor does indeed find insertion sites and would allow further analysis to try to solve the insertion cases.

Figures 8.24 to 8.26 show some examples of the predicted regions for insertions over chromosome 8 of NA12878 with the short read sequencing data used to make the predictions on top. these sequencing reads were generated internally at the Genome Center for the NA12878 sample. The long reads at the bottom come from the GIAB project and are used to assess the validity of the calls. These results were initially used for exploration and experimentation with the development of an insertion caller. This exploration has not been refined into a benchmark and this is one of the reasons why there are no statistics other than the number of predictions made, 1422 calls. The assembled sequences that were shown in chapter 7 are from regions that have been predicted to contain structural variation by this predictor.

From a quick glance at a subset of the results in figures 8.24 to 8.26 we can see that there are not only insertions that come up from the predictor but also deletions and some false positives (no event visible on the aligned long reads). Even if this predictor was not further evaluated or built into a full caller, the regions that are predicted do indeed show signs of variations, as visible on the long reads. Insertions are colored in purple. They are sometimes distributed left and right from the event, this is due to where the alignment algorithm chose to place them in and this is dependent on the read orientation (forward or backward). For example an insertion of "AAAA" in a region that is "AAAAAAAAAAAAA" could be anywhere in the region, the alignment algorithm puts it arbitrarily at one end and this end is opposite when the read is in reverse direction.

Figure 8.24 *Subset of the 1422 variation regions predicted from short reads with long reads at the bottom for reference*

Figure 8.25 *Subset of the 1422 variation regions predicted from short reads with long reads at the bottom for reference*

Figure 8.26 *Subset of the 1422 variation regions predicted from short reads with long reads at the bottom for reference*

Ashkenazim Trio

The Ashkenazim trio (son, father, and mother) was chosen because the Genome in a Bottle consortium (GIAB) has developed benchmark SV calls set for the son (HG002 / NA24385) [191].

The benchmark set was created by merging calls from different tools and sequencing technologies (short-read, long-read, long range, etc.). The set was created from 498'876 candidate insertion and deletion calls over 50 bp from the trio. After removing duplicate calls 296'761 calls remained. When clustering the calls for which the estimated sequence change was less than 20% divergent 128'715 unique SVs remained. The SVs were then further filtered so that those that were confirmed by more than one technology remained as well as the calls that were confirmed by at least 5 call sets from the same technology. This reduced the number of calls to 30'062. These calls were reverified by multiple technologies to keep only the ones very well supported, leaving only 19'748 calls. Finally, calls clustered within 1000 bp of each other were removed, originating from either tandem-repeats or complex regions (hard cases). The final remaining calls formed the benchmark set. The exact process is detailed in [191].

The final benchmark call set enables the assessment of both false negatives and false positives within the benchmark regions. This defines the GIAB Tier 1 benchmark set, which spans 2.66 Gbp and includes 9'641 benchmark SVs (insertions and deletions).

It is important to note that this benchmark reference set of 9'641 SVs (insertions and deletions) comes from more than a 100'000 unique calls as described above. Therefore, false positives may actually be calls for real events that were not kept in the final 9'641 calls. consequently it is important to keep in mind that the true positives and false negatives are reflecting the ability to detect calls that were kept in the benchmark set but that false positives may simply be correct calls for variants that were not kept for the benchmark.

The v0.6 SV benchmark set for HG002 on GRCh37 (hg19) is available at :

`ftp://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/data/AshkenazimTrio/analysis/NIST_SVs_Integration_v0.6/`

Other input SV callsets, assemblies, and other analyses for this trio are available under :

`ftp://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/data/AshkenazimTrio/analysis/`

This reference benchmark data set only contains entries marked as deletions and insertions, no inversions, translocations, or any other events.

8.3.2 HG002 - NA24385

The benchmark calls are only available for the son of the Ashkenazim trio, calls for the parents are also available from multiple sources in the GIAB project but have not been packaged and refined into a benchmark reference call set. There are combined call sets that join all the calls that may be interesting to use for benchmarking, this is however left for future works.

Evaluation of deletions

The deletions were evaluated against the benchmark call set over the whole genome. Since the call set is generated from multiple technologies and not only short reads, it is to be expected that not all events may be detected.

The same caller used for deletion calling on the simulation data set and described in section 4.4.1 was evaluated against the benchmark data set on the whole genome of HG002. The results are summarized in table 8.7.

The performance of the deletion caller on the real data set is much lower than on the simulation data set. Only one fourth to one third of the deletions are correctly called. Benchmarks of the other deletion calling software to which this caller is compared with the simulation data show similar low recall scores in [56] when evaluated on real data sets. The study shows between 20.9% and 28.9% recall scores for the best software. This means that the best of other existing software also tend to only find one fourth to one third of the events. In that study they lowered the reciprocal overlap condition to requiring only 50% overlap. This was also done here but note that the reciprocal overlap condition does not change the results by far. Requiring an 80% overlap only lowers the overall results for precision from 46.4% to 45.1% and the recall from 29.1% to 28.3%. The precision of the best other software is however much better between 74.2% and 91.6%. Although these software were not evaluated on HG002 but on HG001 with different variant calls making a direct comparison impossible they exhibit the same drop in performance when going from simulated data to real data.

The results on the simulated data sets allow to compare the performance between this project and the state of the art, if a further comparison on real data would be needed it would either require running the project on the HG001 data set of the other benchmark or run all the other software with this data set. This is left open for future analysis.

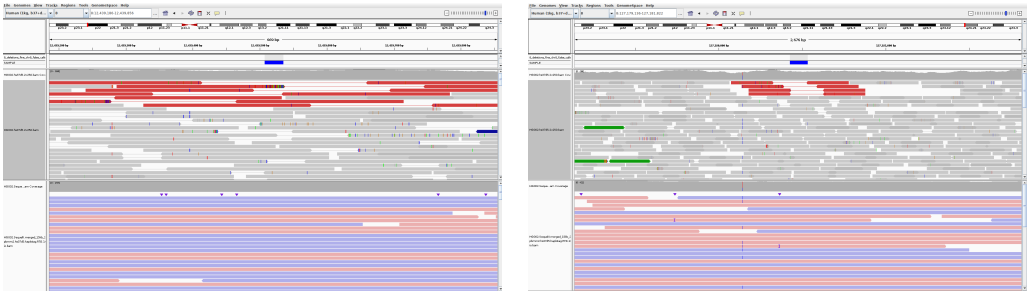
The direct comparison of this deletion caller with the state of the art on the simulation data and indirect comparison on real data sets shows that the deletion caller developed within this project is close to the best of the state of the art but could benefit from improvements and tuning.

One reason for the low precision of the deletion caller on the real data set could be because this dataset was aligned with novoalign instead of BWA or because the sizes of the fragments were not the same. Further evaluation is required in order to fully assess the performance of the caller given the sequencing technology, aligner, and variation type.

Chr	Number of events in truth set	Correctly Predicted (TP)	Missed (FN)	Incorrectly Predicted (FP)	Precision TP / (TP + FP)	Recall TP / (TP + FN)
All	**5464**	**1594**	**3870**	**1841**	**46.4%**	**29.2%**
1	386	120	266	144	45.5%	31.1%
2	450	127	323	141	47.4%	28.2%
3	317	108	209	89	54.8%	34.1%
4	440	146	294	123	54.3%	33.2%
5	343	107	236	95	53.0%	31.2%
6	342	108	234	112	49.1%	31.6%
7	337	97	240	104	48.3%	28.8%
8	328	97	231	89	52.2%	29.6%
9	195	61	134	62	49.6%	31.3%
10	264	76	188	91	45.5%	28.8%
11	233	73	160	91	44.5%	31.3%
12	257	66	191	81	44.9%	25.7%
13	223	67	156	70	48.9%	30.0%
14	166	45	121	58	43.7%	27.1%
15	124	45	79	53	45.9%	36.3%
16	144	24	120	56	30.0%	16.7%
17	187	46	141	74	38.3%	24.6%
18	140	41	99	54	43.2%	29.3%
20	136	36	100	30	54.5%	26.5%
19	179	41	138	47	46.6%	22.9%
22	91	19	72	35	35.2%	20.9%
21	82	17	65	45	27.4%	20.7%
X	98	27	71	66	29.0%	27.6%
Y	2	0	2	31	0.0%	0.0%

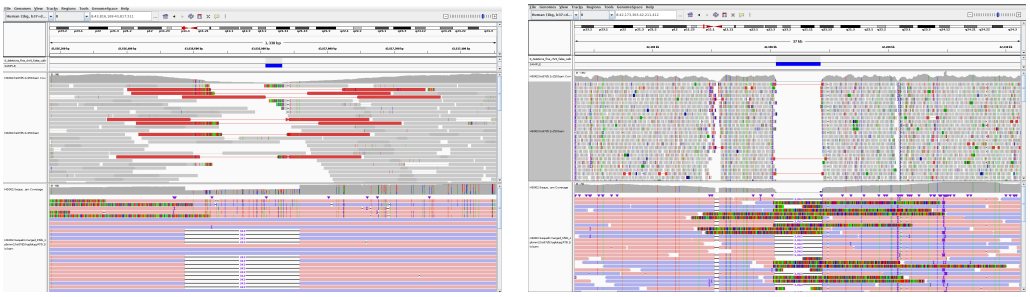
Table 8.7 Global deletion detection results on HG002 from insert size predictor

The false positives, which are the main reason as why the precision score is so low, were explored visually in order to assess the problem. Figures 8.27a to 8.29b show examples of calls marked as false positives. The images are with added long reads on the bottom track in order to further validate the assumptions. Figures 8.27a and 8.27b are real false positives. Figures 8.28a to 8.29b are examples of calls that were for real events but that were not present in the benchmark set.



(a) Real false positive deletion call *(b) Real false positive deletion call*

Figure 8.27 Examples of deletion false positive calls on HG002 chromosome 8



(a) Deletion call *(b) Deletion call*

Figure 8.28 Examples of deletion false positive calls on HG002 chromosome 8 because they were not in the benchmark set

(a) Deletion call *(b) Deletion call*

Figure 8.29 Examples of deletion false positive calls on HG002 chromosome 8 because they were not in the benchmark set

These examples show that false positives can be, and in fact mostly are real events. The advantage of the benchmarking tools and visualization tools is that they can be used in tandem to assess the results. The low precision scores of the caller would be interpreted as bad when looking at the numbers alone. When in reality the calls overlap real events detected in the sequencing data, validated here by short and long read sequencing technologies.

There are of course numerous false positive and false negative calls, nevertheless the benchmarking and visualization tools allow to refine the caller for further improvements.

Finally, in order for a caller to be real effective it should be parameterized based on the exact sequencing and template generation process used to generate the sequencing data, as well as to the aligner used to map the reads back to the reference. With the common problem that is the low availability of benchmark datasets. For example if the caller was improved to maximize the scores of table 8.7 for HG002 based on a single whole-genome sequencing data set, then the caller is very likely to be over-fitted to that particular case and would be generally biased. This is also why it is not necessarily optimal to improve and fine tune predictors at this early stage. The importance for now is that they can run fast and generate calls that can be validated or discarded by human experts. Therefore, it is more important to have a flexible way to create or modify callers than to optimize them on a specific data set. With the approach taken within this project it is possible to use any feature of the sequencing data to make calls and based on the specific experiment or study a predictor or caller can be created in a few hours, the crude predicted results can then be further refined based upon specific analysis needs.

Inter-Chromosomal Fragments

An interchromosomal breakpoint caller created was run on HG002, it applies the algorithm presented in section 4.4.5. This caller generates an output VCF file with breakpoints for every chromosome. The distribution of the mates of the reads that span the breakpoint in terms of chromosome of origin are given for each entry. These entries were finally filtered to keep only relations that are confirmed by more than 10 read pairs. Figures 8.31a and 8.31b show the relations for chromosome 1 and 8 of HG002.

This is an example of another caller that can be implemented using this framework and can give extra insights for more complex structural variation.

Figure 8.30 shows read pairs where one read is mapped on chromosome 8 and the other on chromosome 11. They are marked brown and purple. Reads marked red are reads that have an inferred insert size too big because of the deletion present on chromosome 11.

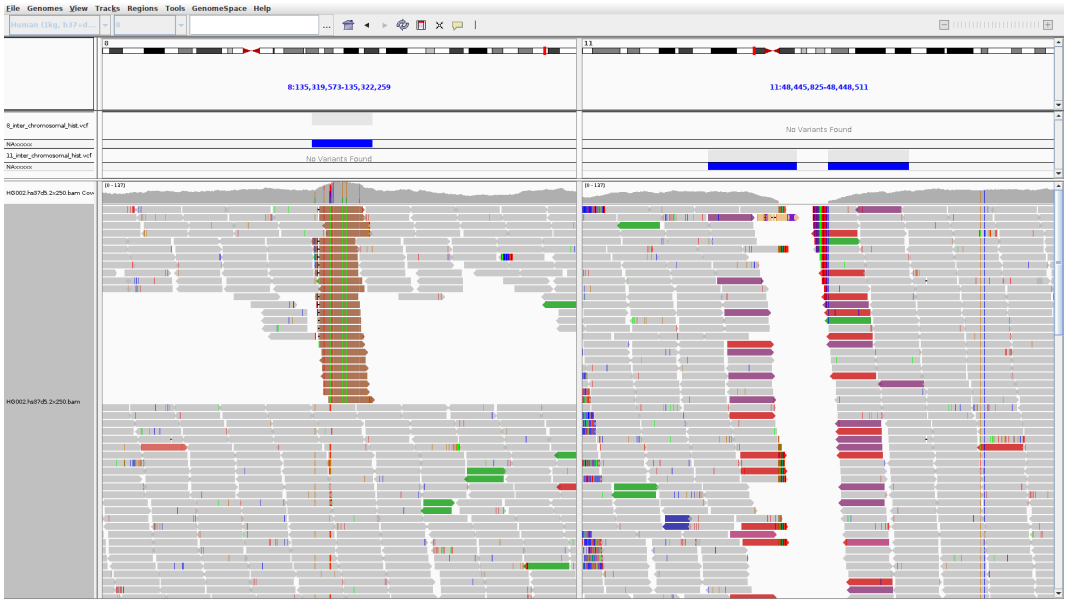
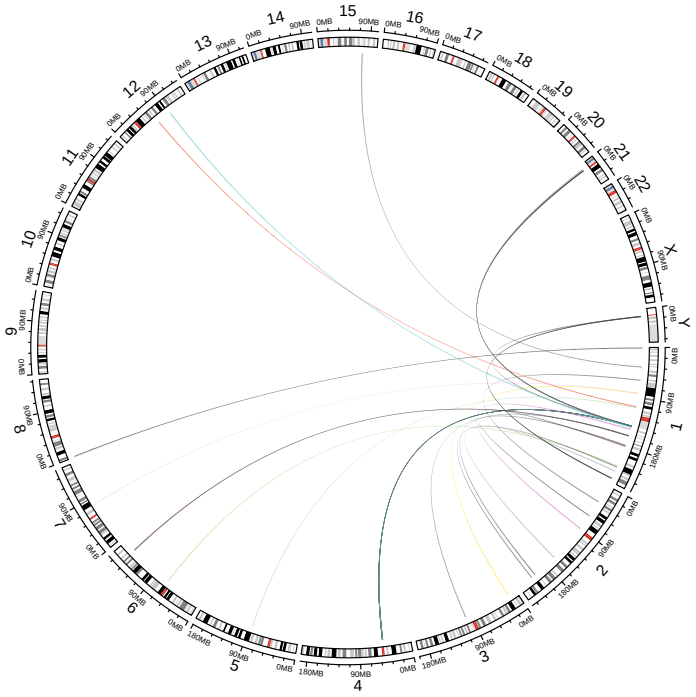
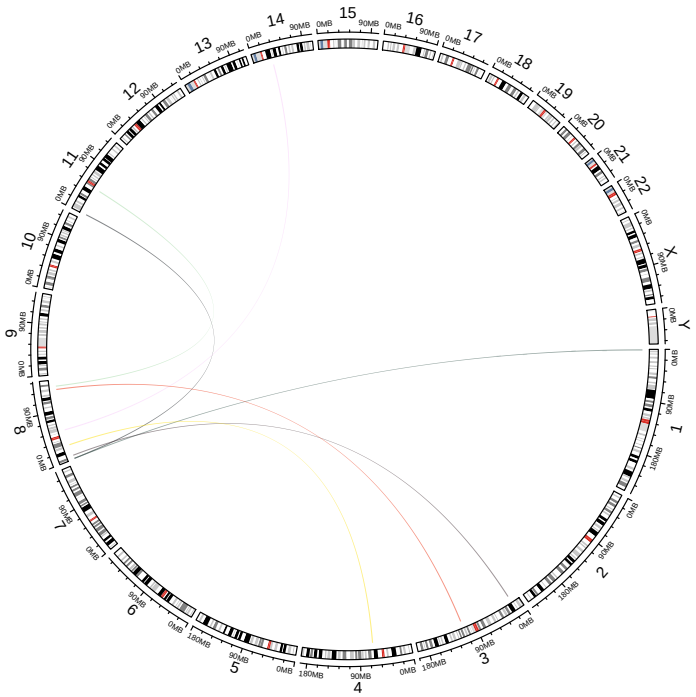


Figure 8.30 *Example of interchromosomal breakpoint, reads on chromosome 8 have their mate mapped on chromosome 11*



(a) Chromosome 1 - Interchromosomal read pair relations



(b) Chromosome 8 - Interchromosomal read pair relations

Figure 8.31 Interchromosomal breakpoints examples found in HG002

Most of these breakpoints are not necessarily the result of a variation between the chromosomes but mostly because reads are mapped wrong (e.g., because a novel insertion sequence maps better to another chromosome than where it happened). In any case, having the reads of a same pair mapped to different chromosomes is a sign that the regions from which the reads originate are not similar to the reference. Therefore, it is sign of structural variation and often contain novel insertions from which the reads originate and map to other chromosomes. Each of theses cases is of interest when studying structural variations and this is an example of another caller that could be developed within the framework.

Conclusion on results on real datasets

Besides the statistics from the benchmark calls on HG002 most of the other calls and predicted breakpoints or regions require further analysis to be conclusive. However, anomalies in the alignment of the reads are often a sign of structural variation and therefore should be predicted. Further studies of these predictions allow to analyze the region and finally classify and call possible structural variations. The results from this projects allow to explore and predict such regions from sequencing data. A more in-depth benchmark is beyond the scope of this thesis and would require more time. Results in terms of runtime performance are introduced below and show that subsequent analysis of multiple datasets will not be a problem in terms of required computing time.

8.4 Runtime Performance

8.4.1 Speed

This section will evaluate the time required to extract signals from whole-genome alignment files as well as for the different predictions and variant calling algorithms.

The machine used to do the tests is comprised of an Intel core i7-4790 (4 cores, 8 threads, up to 4 GHz) CPU, 32 GB of RAM (artificially limited to 8GB) with data on a Samsung 860 QVO SSD drive.

Signal Generation

Signal generation and feature extraction from the alignment file is the most time consuming process. It takes roughly half an hour to extract all signals presented in chapter 7 of whole-genome next generation sequencing data. Extraction time is given in table 8.8 relative to the different datasets.

Data set	Size	Time
Simulation	49.5 GB	24 min 44
HG001	158.2 GB	31 min 55
HG002	130.6 GB	33 min 31

Table 8.8 Time to extract signals from assembly file

This may seem like a long time, but only requires to be done once. Extraction can also be done on a per-signal or per-chromosome basis. Once the signals have been extracted the prediction and calling algorithms can be run on them as often as is required. This saves big amounts of time while developing and tuning the calling algorithms, because the signals and features will already have been extracted and the callers do not require to reprocess the whole alignment file in order to do their predictions.

Calling

Running the deletion, duplication, and inversion callers on the whole genome takes approximately ten minutes for the simulation and HG002 datasets, fifteen minutes were required for the HG001 dataset. Table 8.9 reports the time to run all three callers on the simulation dataset and the real datasets.

Data set	BAM size	Signals size	Calling time
Simulation	49.5 GB	8.3 GB	9 min 15
HG001	158.2 GB	8.3 GB	15 min 40
HG002	130.6 GB	11.3 GB	9 min 57

Table 8.9 Time to call deletions, duplications, and inversions on the whole genome

This means a whole-genome analysis from signal generation to calling can be done in less than one hour on consumer hardware.

Table 8.10 gives the time to run the separate variant callers on chromosome 8 of HG001 with a single core. These results can be compared to the benchmarks in [56]. The benchmarks of that study contain single core runtime for 69 of the state of the art software on chromosome 8 of HG001. Table 8.10 also contains the time required to extract the signals on chromosome 8 with a single core in order to make the comparison more relevant since stand-alone software callers go from BAM file to VCF file, therefore adding the signal generation time with the calling time makes the comparison fair.

Task	Time
Extract all signals	5 min 46 s
Call Deletions	48 s
Call Duplications	1 min 20 s
Call Inversions	1 min 18 s
Predict Insertions	53 s

Table 8.10 Detailed time to call deletions, duplications, inversions, and predict insertion regions on chromosome 8 of HG001

The results show that calling time is relatively low compared to signal generation time. All the signals are extract here. In order to improve the runtime, it would be better to only extract the ones required by the specific callers. Having a low calling time once

the signals are extracted also means that the time required to test a modified or tuned version of the algorithm is very low. Therefore, the development and improvement loop can be very short. This would also be helpful if a machine learning algorithm was used in order to find the best parameters for the algorithms (e.g., threshold values). The same applies when running callers on multiple data sets to benchmark and improve them.

Note on memory usage

The Scala code runs in the Java virtual machine (JVM) and is therefore limited by the amount of memory allocated to the virtual machine. The memory was limited to 8 GB and is therefore not exceeded. Since signal generation, prediction, and calling are done by traversing regions of the genome linearly they do only require a little amount of memory for the current sub-region being handled and this memory can be rapidly recovered when continuing with the next sub-region.

Observed memory usage when performing signal generation was around 1.1 GB (all 10 example signals for whole-genome, parallelized at the chromosome level).

De-novo assembly may require more memory when assembling big regions, the memory usage for assembly should grow linearly with the number of reads used in the assembly.

8.4.2 Disk Usage

Disk usage is mainly dominated by extracted signals, signals that vary more, such as the overall coverage, require more space compared to signals that are stable for large regions thanks to interval encoding. Below is an analysis of generated signals size compared to the initial data set they came from. The initial data set is in BAM (binary, compressed) format, the signals are in bedgraph.gz (bedgraph gzip compressed) format.

The prediction and call files are typically very small and less than a megabyte. Therefore only signal files are listed below.

Simulation data set

The generated extracted signal files amount to a relative size of 16.7% compared to dataset of origin as shown in table 8.11.

Data set size	Signals size	Rel. to data set
49.5 GB	8.3 GB	16.7 %

Table 8.11 Generated signal sizes relative to data set for simulation data set

The signal file sizes do correlate with the length of the chromosome of origin as can be seen in table 8.12 which gives some examples. This linear relationship between size of signals and chromosome size is normal and was expected.

Chr.	Chr. Size	Signals
1	249250621	668.3 MB
2	243199373	709.3 MB
3	198022430	579.8 MB
...		
21	48129895	104.7 MB

Table 8.12 Generated signal sizes relative to chromosomes for simulation data set

Table 8.13 Shows the sizes of individual signal files for the first chromosome. The signals sizes are relative to the number of changes in value they go through in the chromosome. The distribution of file sizes is similar for other chromosomes. It is notable that the coverage signal dominates the total because this signal does change at almost every base, on the contrary signals for unexpected cases such as reads with unmapped mates for example take up very little space because they sit at zero for most of the chromosome and huge regions at a single value massively reduce the file size due to the chosen encoding (valued intervals).

Signal Name	Size	Relative
Read Coverage	482.9 MB	72.3 %
Number of clipped reads	0.9 MB	0.1 %
Number of reads with MAPQ0	11.6 MB	1.7 %
Fragments spanning multiple chromosomes	0.4 MB	<0.1 %
Fragments with discordant tandem orientation	0.06 MB	<0.1 %
Fragments with inferred size too small	3.8 MB	0.6 %
Fragments with inferred size too big	1.4 MB	0.2 %
Sum of edit distance between reads & reference	159.3 MB	23.8 %
Reads with unmapped mates	0.1 MB	<0.1 %
Number of reads that have multiple CIGAR el.	7.8 MB	1.1 %
All	668.3 MB	100 %

Table 8.13 Generated signal sizes for chromosome 1 of simulation data set

HG001 - NA12878

The signal file sizes are detailed in the same way for HG001 in tables 8.14 to 8.16. The most notable difference is that the signals are now less than 10% of the input data size. The total size of all signals is similar to the simulation data set however the sequencing data itself is bigger. This is mostly due to extra information (meta-data) that comes with the reads.

Data set size	Signals size	Rel. to data set
158.2 GB	8.3 GB	5.3 %

Table 8.14 Generated signal sizes relative to data set for HG001 data set

Chr.	Chr. Size	Signals
1	249250621	660.0 MB
2	243199373	694.8 MB
3	198022430	565.3 MB
...		
21	48129895	111.6 MB

Table 8.15 Generated signal sizes relative to chromosomes for HG001 data set

Signal Name	Size	Relative
Read Coverage	462.3 MB	70.1 %
Number of clipped reads	14.0 MB	2.1 %
Number of reads with MAPQ0	12.8 MB	1.9 %
Fragments spanning multiple chromosomes	4.5 MB	0.7 %
Fragments with discordant tandem orientation	0.5 MB	<0.1 %
Fragments with inferred size too small	4.7 MB	0.7 %
Fragments with inferred size too big	1.6 MB	0.2 %
Sum of edit distance between reads & reference	140.2 MB	21.2 %
Reads with unmapped mates	2.0 MB	0.3 %
Number of reads that have multiple CIGAR el.	17.4 MB	2.6 %
All	660.0 MB	100 %

Table 8.16 Generated signal sizes for chromosome 1 of HG001 data set

HG002 - NA24385

The data set for HG002 was aligned with novoalign instead of BWA and the resulting signal data size is a bit higher as can be seen in tables 8.17 and 8.18.

Data set size	Signals size	Rel. to data set
130.6 GB	11.3 GB	8.7 %

Table 8.17 Generated signal sizes relative to data set for HG002 data set

Chr.	Chr. Size	Signals
1	249250621	920.5 MB
2	243199373	957.7 MB
3	198022430	787.4 MB
...		
21	48129895	149.7 MB

Table 8.18 Generated signal sizes relative to chromosomes for HG002 data set

By looking at the details of the individual signal sizes we find out that this is because the signal that represents the sum of the edit distance of the aligned reads is big. Upon investigation it was found out that it was because novoalign has a tendency to clip a few bases on the ends of every read whereas BWA does not. Since clipped bases are considered as an edit in the alignment string to the reference they do affect this signal at almost every position making it much larger in size.

Signal Name	Size	Relative
Read Coverage	539.6 MB	58.6 %
Number of clipped reads	13.2 MB	1.4 %
Number of reads with MAPQ0	0.002 MB	<0.1 %
Fragments spanning multiple chromosomes	0.4 MB	<0.1 %
Fragments with discordant tandem orientation	0.7 MB	<0.1 %
Fragments with inferred size too small	5.1 MB	0.6 %
Fragments with inferred size too big	2.6 MB	0.3 %
Sum of edit distance between reads & reference	319.9 MB	34.8 %
Reads with unmapped mates	8.5 MB	0.9 %
Number of reads that have multiple CIGAR el.	30.5 MB	3.3 %
All	920.5 MB	100 %

Table 8.19 Generated signal sizes for chromosome 1 of HG002 data set

8.4.3 Conclusion on performance

Generating feature signals can be done in roughly half an hour for whole-genome sequencing data of coverage between 30x and 50x. The extra disk space requirement is about 10% of the input data set. These signals then allow to run callers and predictors relatively fast, especially when run on single chromosomes. This approach allows for a faster development cycle compared to "all-in-one" variant callers. Runtime of variant callers was measured in detail in [56] and detailed runtime evaluations of the state of the art software is available in their supplementary materials.

Being able to handle a whole-genome data set in less than one hour on consumer hardware with little memory makes it not only possible to get insights on copy number variants rapidly but also means it would be possible to generate cohort or population studies with a few more resources. The main bottleneck is probably related to bandwidth between the storage disks and the software. Further investigation of the performance in terms of memory and speed would allow to more precisely specify the impact of the hardware, this is left for future works.

9 Discussions

Contents

9.1 Review of the tools developed

During this project a general software framework was developed to help create and benchmark CNV callers. The original goal of the project was to predict the regions that contained CNVs, as precisely as possible, from whole-genome sequencing data. This required the analysis of features in sequencing data that may reflect possible variation events.

A signal extraction tool was developed to extract values that are relevant for variation detection, not only CNVs but all sorts of SVs. This tool allowed to extract features from the assembly data in forms of signals spanning the whole genome. This tool showed to be of invaluable help when designing prediction and calling algorithms not only as input data but also to debug and understand why calls were made. This tool can generate data that can be used for any other subsequent analysis but is also helpful to complement results from other software sources. For example, state of the art software may generate a VCF file but does not necessarily generate anything else. By complementing the calls with the generated signals it is possible to evaluate them on their relation to the signals and have more metrics for analysis.

Within the software framework multiple predictors and variation callers have been developed. The complete tools have been presented either in the tools or in the results chapters, but many more can be created based on need. Incomplete callers such as the insertion caller could not be completely evaluated but can serve as a good basis for a more mature future tool. The extracted signals can be used to predict regions of interest that are probable to contain variation such as insertions. If required, the regions can be further analyzed via algorithms such as de-novo local assembly. This caller for instance was not complete but could already solve cases and show its usefulness.

Interaction with other tools such as Gephi for graph visualization and IGV for more general genomic data visualization shows the possibilities offered by this framework for the development of further tools and improvements on the existing tools. It also provides a way to visually inspect the results of the different callers and algorithms to complement the output statistics.

Benchmarking of results compared to truth data sets allowed to quantify the detection performance of the developed tools and will allow further improvements as well as very precise explanations for the results when complemented with visualization in terms of signal values and aligned reads. This was explored in the results section and made it possible to explain the results in detail.

Overall, the software developed for this project would allow development of not only CNV callers but callers for any type of structural variation. This generic method made it possible to create callers for all type of CNV variations. The resulting variant callers were not necessarily all fully finished tools. Nevertheless it showed that it is possible to create tools that are effective (e.g., Deletions and inversions callers) within this framework. Therefore it seems very likely that more advanced callers such as insertion callers can be refined to mature software tools given the framework.

Rapid evaluation of callers through the framework, requiring only minutes to generate calls over the whole genome and instant benchmarking (done in a few seconds), allow for a very fast development cycle and will make it possible to create better callers in the future.

9.2 Comparison to the state of the art

When comparing the results of the callers to the state of the art, the deletion caller was shown to fall a bit behind but this was expected because the algorithm only used inferred fragment lengths for detection. Improving this algorithm with suggestions shown in the results chapter would probably bring it on the same level as the best tools.

The duplication caller performed poorly mainly with regards to the high number of false positives that the tool did generate. These have been discussed in the results chapter and were shown to be the consequence of incorrectly mapped reads that could be filtered out given the signals created by the signal generation tool.

The inversion calling tool performed as well as the state of the art, mainly because the pair-end short reads are very well suited for detection of inversion events and the events can almost be directly inferred from the sequencing reads.

Sadly, insertions could not be evaluated because of time constraints but preliminary results such as the regions predicted for insertions as well as experimentation with local de-novo assembly tend to indicate that insertions could be detected and sometimes solved with short-reads. Of course, assembly of short reads cannot solve all regions and all novel insertions because of their size, as well as the computational power it would take to do assembly with all reads (because reads from novel insertions could be unmapped or incorrectly mapped elsewhere).

It would still be interesting to only predict regions that could contain insertions and see how well these regions are defined compared to a truth set of insertion cases without looking at the exact genotype. Further analysis of insertions is left for future works.

Further kind of CNVs such as translocations (inter and intrachromosomal) were not evaluated because of the lack of available truth sets.

The results have been investigated by comparing the calls to a truth set of calls, however the benchmarking tools also allow to generate the same results for any set of calls. This means that it would be possible to do a software to software comparison of the call sets and generate the sub call sets in terms of calls that were found by both software, or only one of the two. These "union", "intersection", and "difference" call sets would allow to compare callers for further analysis and help understanding what a specific caller or algorithm can find that another cannot.

9.3 Discussion of the general results

Results of this work are numerous and could not all be evaluated in-depth. This is not necessarily negative, it means that there are numerous possibilities of analysis and further research possible based upon the results of this work. It would be extremely interesting to dive into novel approaches to detect, classify, and solve regions that could contain structural variation and this work makes that possible.

The rather large spectrum of the subject and the choice to explore its multiple aspects did limit the depth of analysis of the individual parts. However it allowed to gain a broad understanding of the concepts around CNV detection and assessment on whole-genome sequencing data.

This project not only created tools for CNV calling but also laid the basis for further exploration of structural variation. The approaches used in the algorithms can be validated, improved, and compared. The results can be interpreted and are backed up by real explanations and not only "numbers". This will allow to develop new tools with relative ease in a good environment.

10 | Materials and Methods

Contents

10.1 Availability of the source code

The source code for this project is available at :

`https://github.com/rick-heig/canevas`

The source code for the VCFotographer tool is available at :

`https://github.com/rick-heig/vcfotographer`

10.1.1 Recreating the results of this work

The instructions to recreate the results will be provided with the source code. Improved and future versions of the algorithms and tools will be updated to this same code repository.

10.2 Availability of the simulated dataset

The simulated dataset can be generated with the following commands :

Clone the git repository from [190] available at

`https://github.com/stat-lab/EvalSVcallers`

The truth set for the variants is available in this repository in the following file :
`Simulated_data_in_our_study/Sim-A/Sim-A.SV.vcf`

A link is given for the FASTA sequence of the simulated sample in the same directory. This FASTA file is then artificially sequenced with the ART software [192].

The following command was used to generate the sequencing files :

```
ART_DIR/art_illumina -sam -i Sim-A.diploid.fa -l 125 -f 15 -m 500 -s 100 -o
↪  Sim-A_30x -p -1 ART_DIR/Illumina_profiles/HiSeq2500L125R1.txt -2
↪  ART_DIR/Illumina_profiles/HiSeq2500L125R2.txt
```

This will use Illumina HiSeq2500 sequencing profiles to generate the simulated paired end reads of length 125 with an average insert size of 500.

The generate FASTQ files where then aligned to the GRCh37 genome with the following command :

```
bwa mem -M -t 8 reference/human_g1k_v37_decoy.fasta Sim-A_30x_b1.fq
↪  Sim-A_30x_b2.fq > Sim-A_30x_g1k_v37.bam
```

The BAM alignment file was then sorted with samtools with the following command :

```
samtools sort Sim-A_30x_g1k_v37.bam -o Sim-A_30x_g1k_v37.sorted.bam
```

and indexed with samtools with the following command :

```
samtools index Sim-A_30x_g1k_v37.sorted.bam
```

10.3 Availability of the real datasets

The real data sets used in this project are available on the GIAB FTP. Except the sequencing reads used for the exploration of insertion sites on NA12878 which were sequenced in-house at the Genome Center.

10.3.1 HG001 - NA12878

All sequencing data :

ftp://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/data/NA12878/

Short reads 30x coverage down sample data (alternative data set to the sequencing reads from the Genome Center with similar coverage) :

ftp://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/data/NA12878/NIST_NA12878_HG001_HiSeq_300x/RMNISTHS_30xdownsample.bam

Long reads used as a reference for possible SVs during exploration :

ftp://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/data/NA12878/NA12878_PacBio_MtSinai/merged_ec_output_primary.bam

10.3.2 HG002 - NA24385

Data set used to evaluate the deletion caller :

ftp://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/data/AshkenazimTrio/HG002_NA24385_son/NIST_Illumina_2x250bps/novoalign_bams/HG002.hs37d5.2x250.bam

Other sequencing data and long reads available at :

ftp://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/data/AshkenazimTrio/HG002_NA24385_son/

Variant calls (benchmark) :

ftp://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/data/AshkenazimTrio/analysis/NIST_SVs_Integration_v0.6/

All variant calls :

ftp://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/data/AshkenazimTrio/analysis/

10.4 Methods

10.4.1 Variant comparison methods

In order to compare two variant call and to be able to analyze if they are close enough to be considered as the same call methods have to be defined. A description of the methods used to tell if two calls are actually identical is given below.

Method for inversions, deletions, and duplications

In order to compare calls from different sources and tell if they are the same variant there needs to be a well defined method. The method used in this work is the same as in the extensive benchmarks [56].

Two calls for a given event (deletion, inversion, insertion) are considered to be the same if their reciprocal overlap is at least 80%. (This condition is reduced to 60% for calls smaller than 1000 bases in length). The reciprocal overlap is shown in figure 10.1.

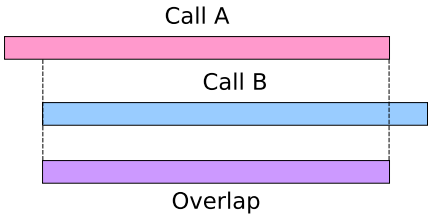


Figure 10.1 Reciprocal overlap between two calls

This means that the overlap between two called regions must represent at least 80% of each call which is equivalent to checking if the intersection of both regions is at least 80% or bigger in size compared to the biggest of both regions.

For real datasets this condition was reduced to a minimal reciprocal overlap of 50%. This was chosen to have the same metric as in [56]. But as discussed in the results chapter reducing the overlap criterion from 80% to 50% had minimal impact on the results.

Note : A more refined method for comparison should include further information such as the copy number. 0 or 1 for deletions depending if the deletion is on one or both alleles. This could also apply to inversions, where it should be stated if they are homozygous or not. For duplications the copy number could also be compared. However, if the goal is only to call the region correctly, this is not a requirement.

Method for insertions

Insertions are often assessed by distance, if two insertions are within a distance below a given threshold they are considered to be the same call. For example a distance of 200 bases was chosen for the insertions benchmarks in [56].

This is enough when the goal is to benchmark the detection of regions that have novel insertion variations. If the calls should also give the genotype of the region and give the sequence of the novel insertion a more involved approach must be taken.

For example, in the creation of the benchmark data set of GIAB for HG002 [191]. insertions were clustered based on location and then merged if the sequence they provided for the insertion was less than 20% divergent.

The method used to benchmark the calls is extremely dependent on what is meant to be shown and should be chosen accordingly.

The benchmark framework of this project can be adapted to integrate any method of comparison. This is a very flexible approach and allows for the creation of multi-level benchmarks, from region prediction to detailed comparison.

10.4.2 Statistical analysis

Calls in the candidate set were accounted as true positives TP calls when they did correspond to a call in the truth set given the comparison methods above. The other calls in the candidate set were accounted as false positives FP. Calls in the truth set that did not have a correspondence in the candidate set were accounted as false negatives FN.

Precision was computed as $Precision = TP/(\text{Candidate calls}) = TP/(TP + FP)$

Recall was computed as $Recall = TP/(\text{True events}) = TP/(TP + FN)$

11 | Conclusion

11.1 Project summary

During the five month time frame of this Master thesis it was possible to get acquainted with the concepts of variation in the genome. To describe the variations known as Copy Number Variations and separate them into categories. Study the signs of variations on aligned reads and analyze them to develop strategies for detection. This led to the development of a collection of software tools written in Scala. This iterative process allowed to explore and try out many strategies and algorithms. This led to the development of several variant callers as well as exploratory tools such as the local de-novo assembler. Finally, the tools were evaluated and benchmarked for performance. The results were not only compared to the state of the art but also explored more in-depth in order to propose strategies and improvements on the current tools.

The large spectrum of this project made it possible to explore numerous aspects of variant detection, classification and resolution. leading to a few finished tools and a large collection of software packages and algorithms that can be useful for the further development of detection and analysis tools.

11.2 Comparison with the initial objectives

The initial objectives were to locate CNVs events by giving breakpoints and delimiting intervals that span the events, as precisely as possible with relation to the position. While classification and resolution were not part of the initial objectives they were implied as exploratory paths during the project.

The results have shown that it was possible to detect different kinds of CNV events by several approaches in order to predict their location based on whole-genome sequencing data. A complete software framework and a collection of tools have been developed that allow to identify relationships between variation events and features of the sequencing data as well as find strategies to automate discovery. This led to the creation of fully functional variation caller tools.

The project also led to feature extraction and predictors that could identify probable regions for every type of CNV, allowing to extract breakpoints, e.g., for insertions, even if the insertion themselves were not solved.

The results have shown that it was possible to build variant callers that were close or on the same level than the state of the art with the framework developed within this project. Callers that did not achieve this level were discussed and the weaknesses of the different approaches have been reported. Exploration of the weaknesses of the current approaches makes it possible to improve these tool based on rational assumptions coming from the original input data and reasoning about the algorithm.

The discussions around these result create multiple follow-up development possibilities and projects as will be described below.

11.3 Future work

Future work includes improving the calling tools. Many hints as to how to do this were given throughout the evaluation of results. This is mostly an iterative process that continues during the whole lifetime of the tools. The tools developed during this project are meant to be used in the Genome Center data analysis and interpretation pipeline, therefore they will continue to evolve and be improved. The simplistic approach taken in some algorithms was also a choice by design that showed that it is possible to create very simple tools that generate results based on decisions that can be understood and traced back from the call to sequencing data. Future works also include adding traceability to the calls, adding information that allows to trace back all the reasons as to why the call was made. This also includes stating the evidence and reasons why some calls were filtered out.

Finishing and improving the insertion caller may also be a major project in the future works. However it may be enough for the moment to only predict regions that could be insertion variants without genotyping the exact sequence.

The results from the callers (CNVs) and predicted regions should also be further evaluated for their relevance with relation to clinical phenotypes and pathologies. These further investigations may be done at the Genome Center. The tools developed within this project will be used to generate call files that will be used for further downstream analysis (E.g., in cohort studies).

The assembly approach with an augmented de Bruijn graph could also lead to interesting results in assembly based analysis of sub-regions of the genome that contain variation. The approach cannot necessarily always solve the regions exactly but the augmented version of the algorithm can provide much more information than the traditional version. This opens up new possibilities such as predicting the length of repetitive regions, predicting if the graph is missing parts (based on the paired read information) or remap subparts of the graph based on location of origin of the k-mers. This can also help to solve more complex SVs such as complex rearrangements.

As for performance improvements, once a specific set of callers has been selected for a project, only the signals that are really needed by those callers have to be generated. Doing this will reduce the time required to generate them as well as the memory footprint of the signals. The callers run fast because the access the data linearly, however some algorithms may benefit from accessing specific parts of the signals based upon a specific need. This random access is not well optimized and could be improved, for example by indexing the file. This is the same idea that is used with SAM/BAM files and the .bai/.sai indexes or for a FASTA file with the .fai index. The algorithms presented within this project did access the signals linearly and not by random access, therefore this did not impact the performance that much, however other algorithms may benefit from this. In most cases the algorithms can be written to access the signals linearly while traversing the region and this is what has been done within this project.

It would also be possible to rewrite the callers in a stand alone manner to directly make the calls from the BAM files without generating the signals. This is easy to do with the framework because of the divide and conquer strategy used in the signal generation.

Therefore a stand alone calling algorithm could be run directly at this stage directly upon signal generation for a sub-region and when finished calling this sub-region the signals can be discarded. This would reduce the extra disk memory requirement to zero. These are consideration for more mature tools. For exploration and development it is much more convenient to have the features already extracted because this reduces the time to run the callers by a great extent.

This concludes the futures works section, leaving many trails open for further exploration. Even if these paths were not explored the software developed will help the researchers examine the different possibilities and guide their way. The numerous suggestions that appear throughout this work may also serve as ideas to further explore the deep field that is structural variation analysis. This work can also be further expanded to other organisms when references are available. If not, there is always the reference-less de-novo approach.

11.4 Final words

This project was very broad and complex. However, with a flexible and iterative methodology, it was possible to explore all of the major aspects of CNV detection and assessment from whole-genome next generation sequencing data. Multiple useful tools have been implemented to guide exploration and analysis of variations. Several variation callers have been implemented and some could already compete with the state of the art software. The computational performance of the tools will also make it possible to apply them to large data set collections. The detected regions and variants will provide new insights on their role and consequences when further studied in downstream analysis.

A | Appendix

A.1 Data formats

A.1.1 FASTA

the FASTA format is a text-based format for representing either nucleotide sequences (DNA / RNA) or amino acid (protein) sequences. The nucleotides or amino acids are represented using single-letter codes and sequences are preceded by an identifier. More information and references can be found on the Wikipedia page which gives a good overview of the format, at :

`https://en.wikipedia.org/wiki/FASTA_format`

A.1.2 FASTQ

The FASTQ format is also text based and very similar to the FASTA format with the major difference that it encodes quality values for sequences. More information and references can be found on the Wikipedia page which gives a good overview of the format, at :

`https://en.wikipedia.org/wiki/FASTQ_format`

A.1.3 SAM / BAM

The SAM format name stands for Sequence Alignment/Map, the BAM is its binary equivalent which comes with a BAI format file for indexing the BAM.

The format specifications are available at :

`https://samtools.github.io/hts-specs/`

More specifically described in the file (v1, which is the current version) :

`http://samtools.github.io/hts-specs/SAMv1.pdf`

A.1.4 VCF

The Variant Call Format VCF is used to store variant calls, the specifications are available at :

`https://samtools.github.io/hts-specs/`

More specifically described in the file (v4.3, which is the current version) :

`http://samtools.github.io/hts-specs/VCFv4.3.pdf`

A.1.5 BED

The BED (Browser Extensible Data) format is a flexible and concise format to represent genomic features and annotations.

It is documented in the UCSC genome browser FAQ here :

`https://genome.ucsc.edu/FAQ/FAQformat.html#format1`

Other descriptions include :

`https://www.ensembl.org/info/website/upload/bed.html`

`https://bedtools.readthedocs.io/en/latest/content/general-usage.html`

A.1.6 Bedgraph

Bedgraph allows to display continuous-valued data over genomic regions, this is very well suited for signals, probability scores, graphs and the like.

The format is described in the UCSC genome browser FAQ here :

`https://genome.ucsc.edu/goldenPath/help/bedgraph.html`

A.1.7 Wig / BigWig

Wig (wiggle) is a format similar to bedgraph and is used for general graphing on genomic regions, wig is an older format and is now replaced by BigWig which is recommended to display dense continuous data on the genome. BigWig is a binary format therefore it may not be very practical to work with directly. BigWig has the advantage of being displayed very rapidly in IGV, the more convenient bedgraph format can be converted to BigWig as described in section A.2.6 if needed for display performances.

The format is described in the UCSC genome browser FAQ here :

`https://genome.ucsc.edu/goldenPath/help/wiggle.html`

A.2 Useful commands

A.2.1 Downloading Files from Genome in a Bottle FTP

Downloading the files on the Genome in a Bottle (GIAB) FTP is possible from a web browser. However when dowloading large files caution should be taken because of possible network interruptions. Downloading the files with a download manager or FTP program allows to resume failed downloads, therefore this is more desirable when handling large files.

```
\begin{Verbatim}
# Example using LFTP - The '-c' is the continue option (aka "reget")
```

```
lftp -e "get -c
↪  ftp://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/data/.../path/to/file; bye"
# Example with links inside a file (bash)
while read link; do lftp -e "get -c ${link}; bye"; done < links.txt
\end{Verbatim}
```

A.2.2 IGV related cookbook

Controlling IGV through a Port

IGV can optionally listen for requests over a port.

See `http://software.broadinstitute.org/software/igv/PortCommands`

```
# Example
echo -n '<command>' | nc <host> <port>
echo -n 'load input.bam' | nc localhost 60151
OK
echo -n 'goto "chr9:132,978,000-132,989,000"' | nc localhost 60151
OK
echo -n 'exit' | nc localhost 60151
```

Note : IGV can sometimes be slow to respond, therefore it can be useful to tell nc to wait a bit before hanging up the connection

```
# Wait for an amount of seconds before hanging up
echo -n '<command>' | nc -q <seconds> <host> <port>
echo -n 'echo' | nc -q 2 localhost 60151
echo
```

A.2.3 FASTA/FASTQ related cookbook

Indexing a FASTA file

This will generate a .fai and possibly .gzi file.

```
# samtools faidx <input file>
samtools faidx bgzip_compressed_input.fa.gz
samtools faidx input.fa
```

Extracting a sequence from an indexed FASTA file

```
samtools faidx reference.fa chr:start-stop
```

A.2.4 Aligning a FASTQ with BWA

First, it is required to generate an index (construct BWT)

```
bwa index reference.fa
```

Second, aligning a FASTQ file can be done with the following command

```
bwa mem reference.fa reads.fq > alignment.bam
```

Alignment of paired-end reads can be done with the following command, the `-p` option is for "smart pairing".

```
bwa mem reference.fa reads_1.fq reads_2.fq > alignment.bam
bwa mem -p reference.fa reads_interleaved_mates.fq > alignment.bam
```

Other interesting options are :

```
bwa mem # To see all options
        -t <threads> "number of threads"
        -M           "mark shorter split hits as secondary"
```

A.2.5 BAM related cookbook

Finding a read

```
samtools view input.bam | grep -m 2 <READ ID>
```

The `-m 2` option is to limit grep to report up to 2 matches. This will avoid grep searching through the whole output of samtools.

Sorting a BAM file

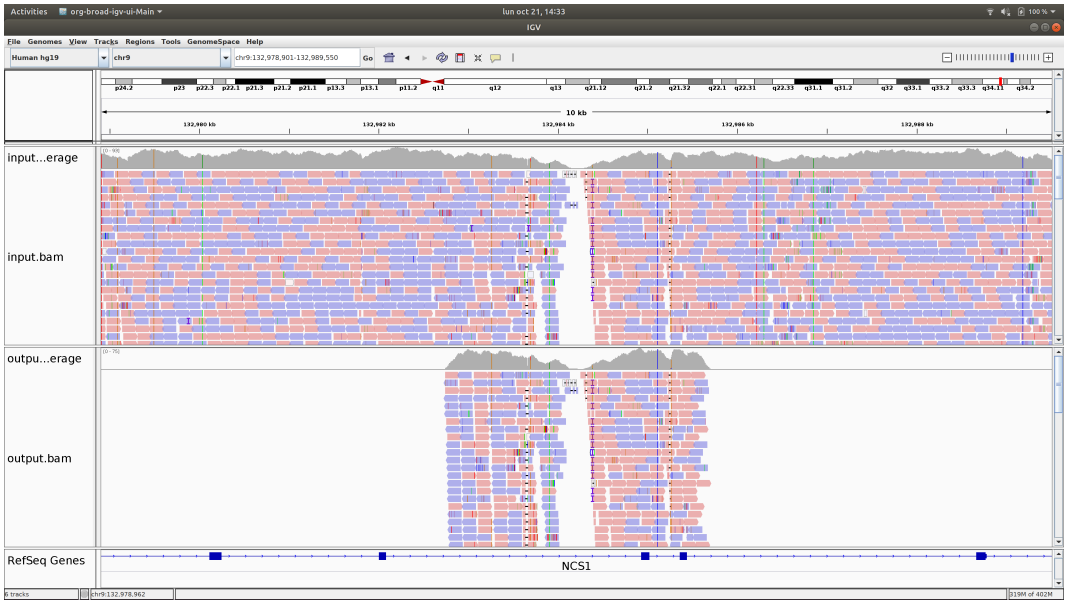
```
samtools sort bamfile.bam -o bamfile.sorted.bam
```

Generating an index (.bai)

```
# With Picard
java -jar /path/to/picard/picard.jar BuildBamIndex I=bamfile.bam
# With Samtools
samtools index bamfile.bam
```

Extracting aligned reads given a region

```
# As SAM
samtools view input.bam "CHR:START-END" > output.sam
# As BAM
samtools view -b input.bam "CHR:START-END" > output.bam
```



Getting read counts with mosdepth

```
docker run -v /path/to/workspace:/opt/mount
↪  quay.io/biocontainers/mosdepth:0.2.6--hfb13731_0 mosdepth --fast-mode -t 4
↪  --by 1000 /opt/mount/output_name /opt/mount/input.bam
```

Mosdepth [193] git repository : `https://github.com/brentp/mosdepth`

Usage :

```
mosdepth 0.2.3

        Usage: mosdepth [options] <prefix> <BAM-or-CRAM>

Arguments:

        <prefix>       outputs: {prefix}.mosdepth.global.dist.txt
                                {prefix}.mosdepth.summary.txt
                                {prefix}.mosdepth.region.dist.txt (if --by is specified)
                                {prefix}.per-base.bed.gz (unless -n/--no-per-base is specified)
                                {prefix}.regions.bed.gz (if --by is specified)
                                {prefix}.quantized.bed.gz (if --quantize is specified)
                                {prefix}.thresholds.bed.gz (if --thresholds is specified)

        <BAM-or-CRAM>  the alignment file for which to calculate depth.

Common Options:

        -t --threads <threads>     number of BAM decompression threads. (use 4 or fewer) [default: 0]
        -c --chrom <chrom>         chromosome to restrict depth calculation.
        -b --by <bed|window>       optional BED file or (integer) window-sizes.
        -n --no-per-base           dont output per-base depth. skipping this output will speed execution
        substantially. prefer quantized or thresholded values if possible.
        -f --fasta <fasta>         fasta file for use with CRAM files.

Other options:

        -F --flag <FLAG>               exclude reads with any of the bits in FLAG set [default: 1796]
        -i --include-flag <FLAG>       only include reads with any of the bits in FLAG set. default is
        ↪  unset. [default: 0]
        -x --fast-mode                 dont look at internal cigar operations or correct mate overlaps
        ↪  (recommended for most use-cases).
        -q --quantize <segments>       write quantized output see docs for description.
        -Q --mapq <mapq>               mapping quality threshold [default: 0]
        -T --thresholds <thresholds>   for each interval in --by, write number of bases covered by at
        least threshold bases. Specify multiple integer values separated
        by ','.
        -R --read-groups <string>      only calculate depth for these comma-separated read groups IDs.
        -h --help                      show help
```

Getting read counts with samtools

Takes a few minutes per chromosome

```
# Will not group region if counts are same as mosdepth does (generates bigger
↪  output file)
samtools depth input.bam > output.txt
# With zeroes
samtools depth -a input.bam > output.txt
```

It is also possible with the `mpileup` samtool option

```
# Check the results (e.g., with less) before saving to file, it will probably
↪  be required to extract desired columns
samtools mpileup -ABQ0 | less
```

Check the samtools manual for the options at :

`https://www.htslib.org/doc/samtools.html`

A.2.6 Other commands

Converting BigWig to Wig

BigWig [194] is a binary format, sometimes it is more convenient to work with the Wig format, to do so a BigWig file can be converted to a Wig file with the following tool.

```
bigWigToWig input.bigwig output.wig
```

Converting bedgraph to BigWig

```
bedGraphToBigWig input.bedgraph chromosome.sizes output.bw
# Note : The chromosome sizes can be a link
bedGraphToBigWig input.bedgraph
↪  https://genome.ucsc.edu/goldenPath/help/hg19.chrom.sizes output.bw
```

The conversion tools can be downloaded from :

`http://hgdownload.soe.ucsc.edu/admin/exe/`

Converting .gz from gzip to bgzip

This may be helpful for samtools which cannot index some files that are compressed with gzip, it requires bgzip compressed files.

```
zcat input.gz | bgzip -c > output.gz
```

References

[1] Joe Hin Tjio and Albert Levan. "The chromosome number of man". In: *Hereditas* 42.1-2 (1956), pp. 1–6.

[2] *Creative Commons : Attribution-ShareAlike 4.0 International (CC BY-SA 4.0)*. URL: `https://creativecommons.org/licenses/by-sa/4.0/deed.en`.

[3] International Human Genome Sequencing Consortium et al. "Initial sequencing and analysis of the human genome". In: *nature* 409.6822 (2001), p. 860.

[4] J Craig Venter et al. "The sequence of the human genome". In: *science* 291.5507 (2001), pp. 1304–1351.

[5] Donald F Conrad et al. "Origins and functional impact of copy number variation in the human genome". In: *Nature* 464.7289 (2010), p. 704.

[6] Peter H Sudmant et al. "Diversity of human copy number variation and multicopy genes". In: *Science* 330.6004 (2010), pp. 641–646.

[7] Jennifer L Freeman et al. "Copy number variation: new insights in genome diversity". In: *Genome research* 16.8 (2006), pp. 949–961.

[8] Ryan E Mills et al. "Mapping copy number variation by population-scale genome sequencing". In: *Nature* 470.7332 (2011), p. 59.

[9] Peter H Sudmant et al. "An integrated map of structural variation in 2,504 human genomes". In: *Nature* 526.7571 (2015), p. 75.

[10] Peter H Sudmant et al. "Global diversity, population stratification, and selection of human copy-number variation". In: *Science* 349.6253 (2015), aab3761.

[11] Joachim Weischenfeldt et al. "Phenotypic impact of genomic structural variation: insights from and for human disease". In: *Nature Reviews Genetics* 14.2 (2013), p. 125.

[12] Malte Spielmann, Darıo G Lupiáñez, and Stefan Mundlos. "Structural variation in the 3D genome". In: *Nature Reviews Genetics* 19.7 (2018), p. 453.

[13] Aaron McKenna et al. "The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data". In: *Genome research* 20.9 (2010), pp. 1297–1303.

[14] Larry Gonick and Mark Wheelis. *The Cartoon Guide to Genetics*. Harper Perennial, 1991.

[15] Mikael Haggstrom et al. "Medical gallery of Mikael Haggstrom 2014". In: *WikiJournal of Medicine* 1.2 (2014), p. 1.

[16] Shuichi Hoshika et al. "Hachimoji DNA and RNA: A genetic system with eight building blocks". In: *Science* 363.6429 (2019), pp. 884–887.

[17] Juliann Chmielecki and Matthew Meyerson. "DNA sequencing of cancer: what have we learned?" In: *Annual review of medicine* 65 (2014), pp. 63–79.

[18] Adam R Abate et al. "DNA sequence analysis with droplet-based microfluidics". In: *Lab on a Chip* 13.24 (2013), pp. 4864–4869.

[19] J Larry Jameson and Dan L Longo. "Precision medicine—personalized, problematic, and promising". In: *Obstetrical & gynecological survey* 70.10 (2015), pp. 612–614.

[20] Kevin Davies. *The $1,000 genome: the revolution in DNA sequencing and the new era of personalized medicine*. Simon and Schuster, 2015.

[21] JY Gall Le and Patrice Debre. "Genome sequencing and personalized medicine: perspectives and limitations". In: *Bulletin de l'Academie nationale de medecine* 198.1 (2014), pp. 101–117.

[22] C William Birky Jr et al. "Using population genetic theory and DNA sequences for species detection and identification in asexual organisms". In: *PLoS one* 5.5 (2010), e10609.

[23] C William Birky Jr. "Species detection and identification in sexual organisms using population genetic theory and DNA sequences". In: *PLoS One* 8.1 (2013), e52544.

[24] Yunan Luo et al. "Metagenomic binning through low-density hashing". In: *Bioinformatics* 35.2 (2018), pp. 219–226.

[25] Brian D Ondov et al. "Mash: fast genome and metagenome distance estimation using MinHash". In: *Genome biology* 17.1 (2016), p. 132.

[26] *The Human Genome Project*. URL: `https://www.genome.gov/human-genome-project`.

[27] Jeremy Schmutz et al. "Quality assessment of the human genome sequence". In: *Nature* 429.6990 (2004), p. 365.

[28] *National Human Genome Research Institute - DNA Sequencing Costs: Data*. URL: `https://www.genome.gov/about-genomics/fact-sheets/DNA-Sequencing-Costs-Data`.

[29] Fred Sanger and Alan R Coulson. "A rapid method for determining sequences in DNA by primed synthesis with DNA polymerase". In: *Journal of molecular biology* 94.3 (1975), pp. 441–448.

[30] Frederick Sanger, Steven Nicklen, and Alan R Coulson. "DNA sequencing with chain-terminating inhibitors". In: *Proceedings of the national academy of sciences* 74.12 (1977), pp. 5463–5467.

[31] *File:Sanger-sequencing.svg*. URL: `https://commons.wikimedia.org/wiki/File:Sanger-sequencing.svg`.

[32] *Creative Commons : Attribution-ShareAlike 3.0 Unported (CC BY-SA 3.0)*. URL: `https://creativecommons.org/licenses/by-sa/3.0/deed.en`.

[33] Simon Ardui et al. "Single molecule real-time (SMRT) sequencing comes of age: applications and utilities for medical diagnostics". In: *Nucleic acids research* 46.5 (2018), pp. 2159–2168.

[34] Raga Krishnakumar et al. "Systematic and stochastic influences on the performance of the MinION nanopore sequencer across a range of nucleotide bias". In: *Scientific reports* 8.1 (2018), pp. 1–13.

[35] Aaron M Wenger et al. "Highly-accurate long-read sequencing improves variant detection and assembly of a human genome". In: *BioRxiv* (2019), p. 519025.

[36] Shanika L Amarasinghe et al. "Opportunities and challenges in long-read sequencing data analysis". In: *Genome Biology* 21.1 (2020), pp. 1–16.

[37] Karen H Miga et al. "Centromere reference models for human chromosomes X and Y satellite arrays". In: *Genome research* 24.4 (2014), pp. 697–707.

[38] James T Robinson et al. "Integrative genomics viewer". In: *Nature biotechnology* 29.1 (2011), p. 24.

[39] Helga Thorvaldsdóttir, James T Robinson, and Jill P Mesirov. "Integrative Genomics Viewer (IGV): high-performance genomics data visualization and exploration". In: *Briefings in bioinformatics* 14.2 (2013), pp. 178–192.

[40] James T Robinson et al. "Variant review with the integrative genomics viewer". In: *Cancer research* 77.21 (2017), e31–e34.

[41] *Integratie Genomics Viewer*. URL: `https://software.broadinstitute.org/software/igv/` (visited on 01/13/2020).

[42] *Genome in a Bottle*. URL: `https://www.nist.gov/programs-projects/genome-bottle`.

[43] International HapMap Consortium et al. "The international HapMap project". In: *Nature* 426.6968 (2003), p. 789.

[44] George M Church. "The personal genome project". In: *Molecular systems biology* 1.1 (2005). URL: `https://www.personalgenomes.org/us`.

[45] Peter Krusche et al. "Best practices for benchmarking germline small-variant calls in human genomes". In: *Nature biotechnology* 37.5 (2019), p. 555. URL: `https://github.com/ga4gh/benchmarking-tools/`.

[46] Ryan Poplin et al. "Scaling accurate genetic variant discovery to tens of thousands of samples". In: *BioRxiv* (2018), p. 201178.

[47] Steve S. Ho, Alexander E. Urban, and Ryan E. Mills. "Structural variation in the sequencing era". In: *Nature Reviews Genetics* (2019).

[48] Ryan M Layer et al. "LUMPY: a probabilistic framework for structural variant discovery". In: *Genome biology* 15.6 (2014), R84. URL: `https://github.com/arq5x/lumpy-sv`.

[49] Marghoob Mohiyuddin et al. "MetaSV: an accurate and integrative structural-variant caller for next generation sequencing". In: *Bioinformatics* 31.16 (2015), pp. 2741–2744. URL: `https://github.com/bioinform/metasv`.

[50] Vijay Kumar Pounraja et al. "A machine-learning approach for accurate detection of copy number variants from exome sequencing". In: *Genome research* 29.7 (2019), pp. 1134–1143.

[51] *dbVar - NCBI's database of human genomic Structural Variation*. URL: `https://www.ncbi.nlm.nih.gov/dbvar`.

[52] *ClinVar*. URL: `https://www.ncbi.nlm.nih.gov/clinvar/`.

[53] *Database of Genomic Variants*. URL: `http://dgv.tcag.ca/dgv/app/home`.

[54] *Database of Genomic Variants archive*. URL: `https://www.ebi.ac.uk/dgva/`.

[55] *European Variation Archive*. URL: https://www.ebi.ac.uk/eva/ (visited on 01/17/2020).

[56] Shunichi Kosugi et al. "Comprehensive evaluation of structural variation detection algorithms for whole genome sequencing". In: *Genome biology* 20.1 (2019), p. 117.

[57] *1-2-3-SV*. URL: https://github.com/Vityay/1-2-3-SV.

[58] Mehrdad Bakhtiari et al. "Targeted genotyping of variable number tandem repeats with adVNTR". In: *Genome research* 28.11 (2018), pp. 1709–1719. URL: https://github.com/mehrdadbakhtiari/adVNTR.

[59] WeiBo Wang et al. "Allele-specific copy-number discovery from whole-genome and whole-exome sequencing". In: *Nucleic acids research* 43.14 (2015), e90–e90. URL: https://sourceforge.net/projects/asgenseng/.

[60] Manuel Holtgrewe, Leon Kuchenbecker, and Knut Reinert. "Methods for the detection and assembly of novel sequence in high-throughput sequencing data". In: *Bioinformatics* 31.12 (2015), pp. 1904–1912. URL: https://github.com/seqan/anise_basil.

[61] Chandana Tennakoon and Wing Kin Sung. "BATVI: fast, sensitive and accurate detection of virus integrations". In: *BMC bioinformatics* 18.3 (2017), p. 71. URL: https://github.com/22noon/BatVI.

[62] Ruibin Xi et al. "Copy number analysis of whole-genome data using BIC-seq2 and its application to detection of cancer susceptibility variants". In: *Nucleic acids research* 44.13 (2016), pp. 6274–6286. URL: https://github.com/ding-lab/BICSEQ2.

[63] *Bionano Solve Theory of Operation: Structural Variant Calling*. 30110 Rev. F. Bionano Genomics. 2019.

[64] *Guidelines for Running Bionano Solve on the Command Line*. 30205 Rev. F. Bionano Genomics. 2019.

[65] *Introduction to Copy Number Analysis*. 30210 Rev. D. Bionano Genomics. 2019.

[66] Ken Chen et al. "BreakDancer: an algorithm for high-resolution mapping of genomic structural variation". In: *Nature methods* 6.9 (2009), p. 677. URL: https://github.com/genome/breakdancer.

[67] Xian Fan et al. "BreakDancer: Identification of genomic structural variation from paired-end read mapping". In: *Current protocols in bioinformatics* 45.1 (2014), pp. 15–6. URL: https://github.com/genome/breakdancer.

[68] Hui Zhao and Fangqing Zhao. "BreakSeek: a breakpoint-based algorithm for full spectral range INDEL detection". In: *Nucleic acids research* 43.14 (2015), pp. 6701–6713. URL: https://sourceforge.net/projects/breakseek/.

[69] Hugo YK Lam et al. "Nucleotide-resolution analysis of structural variants using BreakSeq and a breakpoint library". In: *Nature biotechnology* 28.1 (2010), p. 47. URL: https://github.com/bioinform/breakseq2.

[70] Alexej Abyzov et al. "Analysis of deletion breakpoints from 1,092 humans reveals details of mutation mechanisms". In: *Nature communications* 6.1 (2015), pp. 1–12.

[71] Michael James Clark et al. "U87MG decoded: the genomic sequence of a cytogenetically aberrant human cancer cell line". In: *PLoS genetics* 6.1 (2010). URL: https://sourceforge.net/projects/breakway/.

[72] Eric Roller et al. "Canvas: versatile and scalable detection of copy number variants". In: *Bioinformatics* 32.15 (2016), pp. 2375–2377. URL: https://github.com/Illumina/canvas.

[73] Sergii Ivakhno et al. "Canvas SPW: calling de novo copy number variants in pedigrees". In: *Bioinformatics* 34.3 (2017), pp. 516–518.

[74] Tobias Marschall et al. "CLEVER: clique-enumerating variant finder". In: *Bioinformatics* 28.22 (2012), pp. 2875–2882. URL: https://bitbucket.org/tobiasmarschall/clever-toolkit/.

[75] Günter Klambauer et al. "cn. MOPS: mixture of Poissons for discovering copy number variations in next-generation sequencing data with a low false discovery rate". In: *Nucleic acids research* 40.9 (2012), e69–e69. URL: http://www.bioinf.jku.at/software/cnmops/cnmops.html.

[76] Zhanyong Wang et al. "CNVeM: copy number variation detection using uncertainty of read mapping". In: *Journal of Computational Biology* 20.3 (2013), pp. 224–236. URL: https://sph.umich.edu/csg/software_pages/cnvem.html.

[77] Paul Medvedev et al. "Detecting copy number variation with mated short reads". In: *Genome research* 20.11 (2010), pp. 1613–1622. URL: http://compbio.cs.toronto.edu/CNVer/.

[78] Alexej Abyzov et al. "CNVnator: an approach to discover, genotype, and characterize typical and atypical CNVs from family and population genome sequencing". In: *Genome research* 21.6 (2011), pp. 974–984. URL: https://github.com/abyzovlab/CNVnator.

[79] Hoang T Nguyen, Tony R Merriman, and Michael A Black. "The CNVrd2 package: measurement of copy number at complex loci using high-throughput sequencing data". In: *Frontiers in Genetics* 5 (2014), p. 248. URL: https://github.com/hoangtn/CNVrd2.

[80] Hoang Tan Nguyen, Tony R Merriman, and Michael A Black. "CNVrd2: A package for measuring gene copy number, identifying SNPs tagging copy number variants, and detecting copy number polymorphic genomic regions". In: (2016). URL: https://github.com/hoangtn/CNVrd2.

[81] Yuchao Jiang et al. "CODEX2: full-spectrum copy number variation detection by high-throughput DNA sequencing". In: *Genome biology* 19.1 (2018), p. 202. URL: https://github.com/yuchaojiang/CODEX2.

[82] Valentina Boeva et al. "Control-FREEC: a tool for assessing copy number and allelic content using next-generation sequencing data". In: *Bioinformatics* 28.3 (2012), pp. 423–425. URL: https://github.com/BoevaLab/FREEC.

[83] Lennart F Johansson et al. "CoNVaDING: single exon variation detection in targeted NGS data". In: *Human mutation* 37.5 (2016), pp. 457–464. URL: https://github.com/molgenis/CoNVaDING.

[84] Zachary Stephens et al. "Detection and visualization of complex structural variants from long reads". In: *BMC bioinformatics* 19.20 (2018), p. 508. URL: https://github.com/zstephens/CORGi.

[85] Jianmin Wang et al. "CREST maps somatic structural variation in cancer genomes with base-pair resolution". In: *Nature methods* 8.8 (2011), pp. 652–654. URL: https://github.com/youngmook/CREST.

[86] Tobias Rausch et al. "DELLY: structural variant discovery by integrated paired-end and split-read analysis". In: *Bioinformatics* 28.18 (2012), pp. i333–i339. URL: https://github.com/dellytools/delly.

[87] Jana Ebler, Alexander Schönhuth, and Tobias Marschall. "Genotyping inversions and tandem duplications". In: *Bioinformatics* 33.24 (2017), pp. 4015–4023. URL: https://bitbucket.org/jana_ebler/digtyper.git.

[88] Gargi Dayama et al. "The genomic landscape of polymorphic human nuclear mitochondrial insertions". In: *Nucleic acids research* 42.20 (2014), pp. 12640–12649. URL: https://github.com/mills-lab/dinumt.

[89] Mingfu Zhu et al. "Using ERDS to infer copy-number variants in high-coverage genomes". In: *The American Journal of Human Genetics* 91.3 (2012), pp. 408–421. URL: https://github.com/igm-team/ERDS.

[90] Heng Li. "FermiKit: assembly-based variant calling for Illumina resequencing data". In: *Bioinformatics* 31.22 (2015), pp. 3694–3696. URL: https://github.com/lh3/fermikit.

[91] Jacob J Michaelson and Jonathan Sebat. "forestSV: structural variant discovery through statistical learning". In: *Nature methods* 9.8 (2012), p. 819.

[92] Timothy Becker et al. "FusorSV: an algorithm for optimally combining data from multiple structural variation detection methods". In: *Genome biology* 19.1 (2018), p. 38. URL: https://github.com/timothyjamesbecker/FusorSV.

[93] Suzanne S Sindi et al. "An integrative probabilistic model for identification of structural variation in sequencing data". In: *Genome biology* 13.3 (2012), R22. URL: http://compbio.cs.brown.edu/projects/gasv/.

[94] Stephanie U Greer and Hanlee P Ji. "Structural variant analysis for linked-read sequencing data with gemtools". In: *Bioinformatics* (2019). URL: https://github.com/sgreer77/gemtools.

[95] Robert E Handsaker et al. "Discovery and genotyping of genome structural polymorphism by sequencing on a population scale". In: *Nature genetics* 43.3 (2011), p. 269. URL: https://software.broadinstitute.org/software/genomestrip/.

[96] Robert E Handsaker et al. "Large multiallelic copy number variations in humans". In: *Nature genetics* 47.3 (2015), p. 296. URL: https://software.broadinstitute.org/software/genomestrip/.

[97] Daniel L Cameron et al. "GRIDSS: sensitive and specific genomic rearrangement detection using positional de Bruijn graph assembly". In: *Genome research* 27.12 (2017), pp. 2050–2060. URL: https://github.com/PapenfussLab/GRIDSS.

[98] Noah Spies et al. "Genome-wide reconstruction of complex structural variants using read clouds". In: *Nature methods* 14.9 (2017), p. 915. URL: https://github.com/grocsvs/grocsvs.

[99] Sean D Smith, Joseph K Kawash, and Andrey Grigoriev. "GROM-RD: resolving genomic biases to improve read depth detection of copy number variants". In: *PeerJ* 3 (2015), e836. URL: http://grigoriev.rutgers.edu/software/grom-rd/index.html.

[100] Saurabh Baheti et al. "HGT-ID: an efficient and sensitive workflow to detect human-viral insertion sites using next-generation sequencing data". In: *BMC bioinformatics* 19.1 (2018), p. 271. URL: http://kalarikrlab.org/Software/HGT-ID.html.

[101] Hugo YK Lam et al. "Detecting and annotating genetic variations using the HugeSeq pipeline". In: *Nature biotechnology* 30.3 (2012), p. 226. URL: https://github.com/StanfordBioinformatics/HugeSeq.

[102] Aaron R Quinlan et al. "Genome-wide mapping and assembly of structural variant breakpoints in the mouse genome". In: *Genome research* 20.5 (2010), pp. 623–635. URL: https://github.com/arq5x/Hydra.

[103] Xian Fan et al. "HySA: a Hybrid Structural variant Assembly approach using next-generation and single-molecule sequencing technologies". In: *Genome research* 27.5 (2017), pp. 793–800. URL: https://bitbucket.org/xianfan/hybridassemblysv.

[104] Prashanthi Dharanipragada, Sriharsha Vogeti, and Nita Parekh. "iCopyDAV: Integrated platform for copy number variations—Detection, annotation and visualization". In: *PloS one* 13.4 (2018). URL: https://github.com/vogetihrsh/icopydav.

[105] Takahiro Mimori et al. "iSVP: an integrated structural variant calling pipeline from high-throughput sequencing data". In: *BMC systems biology* 7.6 (2013), S8.

[106] Aakrosh Ratan et al. "Identification of indels in next-generation sequencing data". In: *BMC bioinformatics* 16.1 (2015), p. 42. URL: www.bx.psu.edu/miller_lab/indelMINER.tar.gz.

[107] Ji Qi and Fangqing Zhao. "inGAP-sv: a novel scheme to identify and visualize structural variation from paired end mapping data". In: *Nucleic acids research* 39.suppl_2 (2011), W567–W575. URL: http://ingap.sourceforge.net/.

[108] Chuan Jiang et al. "ITIS, a bioinformatics tool for accurate identification of transposon insertion sites using next-generation sequencing data". In: *BMC bioinformatics* 16.1 (2015), p. 72.

[109] Jiali Zhuang and Zhiping Weng. "Local sequence assembly reveals a high-resolution profile of somatic structural variations in 97 cancer genomes". In: *Nucleic acids research* 43.17 (2015), pp. 8146–8156. URL: https://github.com/JialiUMassWengLab/laSV/.

[110] Li Fang et al. "LinkedSV for detection of mosaic structural variants from linked-read exome and genome sequencing data". In: *Nature communications* 10.1 (2019), pp. 1–15. URL: https://github.com/WGLab/LinkedSV.

[111] Martin C Frith and Sofia Khan. "A survey of localized sequence rearrangements in human DNA". In: *Nucleic acids research* 46.4 (2017), pp. 1661–1673. URL: https://github.com/mcfrith/local-rearrangements.

[112] Grace XY Zheng et al. "Haplotyping germline and cancer genomes with high-throughput linked-read sequencing". In: *Nature biotechnology* 34.3 (2016), p. 303. URL: https://github.com/10XGenomics/longranger.

[113] Xiaoyu Chen et al. "Manta: rapid detection of structural variants and indels for clinical sequencing applications". In: *bioRxiv* (2015), p. 024232. URL: https://github.com/Illumina/manta.

[114] Xiaoyu Chen et al. "Manta: rapid detection of structural variants and indels for germline and cancer sequencing applications". In: *Bioinformatics* 32.8 (2016), pp. 1220–1222. URL: https://github.com/Illumina/manta.

[115] Yinghua Wu et al. "MATCHCLIP: locate precise breakpoints for copy number variation using CIGAR string by matching soft clipped reads". In: *Frontiers in genetics* 4 (2013), p. 157. URL: https://github.com/yhwu/matchclips/.

[116] Lixing Yang et al. "Diverse mechanisms of somatic structural variations in human cancer genomes". In: *Cell* 153.4 (2013), pp. 919–929. URL: http://compbio.med.harvard.edu/Meerkat/.

[117] Eugene J Gardner et al. "The Mobile Element Locator Tool (MELT): population-scale mobile element discovery and biology". In: *Genome research* 27.11 (2017), pp. 1916–1929. URL: https://melt.igs.umaryland.edu.

[118] Guillaume Rizk et al. "MindTheGap: integrated detection and assembly of short and long insertions". In: *Bioinformatics* 30.24 (2014), pp. 3451–3457. URL: https://github.com/GATB/MindTheGap.

[119] Djie Tjwan Thung et al. "Mobster: accurate detection of mobile element insertions in next generation sequencing data". In: *Genome biology* 15.10 (2014), p. 488. URL: https://sourceforge.net/projects/mobster/.

[120] Can Alkan et al. "Personalized copy number and segmental duplication maps using next-generation sequencing". In: *Nature genetics* 41.10 (2009), p. 1061. URL: http://mrcanavar.sourceforge.net.

[121] Anna Ritz et al. "Characterization of structural variants with single molecule and hybrid sequencing approaches". In: *Bioinformatics* 30.24 (2014), pp. 3458–3466. URL: https://github.com/raphael-group/multibreak-sv.

[122] Rebecca Elyanow, Hsin-Ta Wu, and Benjamin J Raphael. "Identifying structural variants using linked-read sequencing data". In: *Bioinformatics* 34.2 (2017), pp. 353–360. URL: https://github.com/raphael-group/NAIBR.

[123] Mircea Cretu Stancu et al. "Mapping and phasing of structural variation in patient genomes using nanopore sequencing". In: *Nature communications* 8.1 (2017), p. 1326. URL: https://github.com/mroosmalen/nanosv.

[124] Li Fang et al. "NextSV: a meta-caller for structural variants from low-coverage long-read sequencing data". In: *BMC bioinformatics* 19.1 (2018), p. 180. URL: https://github.com/Nextomics/nextsv.

[125] Dmitry Meleshko et al. "Detection and assembly of novel sequence insertions using Linked-Read technology". In: *bioRxiv* (2019), p. 551028. URL: https://github.com/1dayac/Novel-X.

[126] Haojing Shao et al. "npInv: accurate detection and genotyping of inversions using long read sub-alignment". In: *BMC Bioinformatics* 19.1 (2018), p. 261. ISSN: 1471-2105. URL: https://github.com/haojingshao/npInv.

[127] Le Li et al. "OMSV enables accurate and comprehensive identification of large structural variations from nanochannel-based single-molecule optical maps". In: *Genome biology* 18.1 (2017), p. 230. URL: https://github.com/moziya/OMSV.

[128] Christopher Yau. "OncoSNP-SEQ: a statistical approach for the identification of somatic copy number alterations from next-generation sequencing of cancer genomes". In: *Bioinformatics* 29.19 (2013), pp. 2482–2484. URL: https://sites.google.com/site/oncosnpseq/.

[129] Ajay Ummat and Ali Bashir. "Resolving complex tandem repeats with long reads". In: *Bioinformatics* 30.24 (2014), pp. 3491–3498. URL: https://github.com/alibashir/pacmonstr.

[130] Weichen Zhou et al. "Identification and characterization of occult human-specific LINE-1 insertions using long-read sequencing technology". In: *Nucleic Acids Research* (2019). URL: https://github.com/mills-lab/PALMER.

[131] Pınar Kavak et al. "Discovery and genotyping of novel sequence insertions in many sequenced individuals". In: *Bioinformatics* 33.14 (2017), pp. i161–i169. URL: https://github.com/vpc-ccg/pamir.

[132] Samantha Zarate et al. "Parliament2: Fast structural variant calling using optimized combinations of callers". In: *BioRxiv* (2018), p. 424267. URL: https://github.com/dnanexus/parliament2.

[133] Adam C English, William J Salerno, and Jeffrey G Reid. "PBHoney: identifying genomic variants via long-read discordance and interrupted mapping". In: *BMC bioinformatics* 15.1 (2014), p. 180. URL: https://sourceforge.net/projects/pb-jelly/.

[134] *pbsv, PacBio structural variant (SV) calling and analysis tools*. https://github.com/PacificBiosciences/pbsv. 2019.

[135] Kai Wang et al. "PennCNV: an integrated hidden Markov model designed for high-resolution copy number variation detection in whole-genome SNP genotyping data". In: *Genome research* 17.11 (2007), pp. 1665–1674. URL: https://github.com/WGLab/PennCNV.

[136] Liang Gong et al. "Picky comprehensively detects high-resolution structural variants in nanopore long reads". In: *Nature methods* 15.6 (2018), p. 455. URL: https://github.com/TheJacksonLaboratory/Picky.

[137] Kai Ye et al. "Pindel: a pattern growth approach to detect break points of large deletions and medium sized insertions from paired-end short reads". In: *Bioinformatics* 25.21 (2009), pp. 2865–2871. URL: https://github.com/genome/pindel.

[138] Birte Kehr, Páll Melsted, and Bjarni V Halldórsson. "PopIns: population-scale detection of novel sequence insertions". In: *Bioinformatics* 32.7 (2016), pp. 961–967. URL: https://github.com/bkehr/popins.

[139] Yue Jiang, Yadong Wang, and Michael Brudno. "PRISM: pair-read informed split-read mapping for base-pair level detection of insertion, deletion and structural variants". In: *Bioinformatics* 28.20 (2012), pp. 2576–2583. URL: http://compbio.cs.toronto.edu/prism.

[140] Derek M Bickhart et al. "RAPTR-SV: a hybrid method for the detection of structural variants". In: *Bioinformatics* 31.13 (2015), pp. 2084–2090. URL: https://github.com/njdbickhart/RAPTR-SV.

[141] Christopher A Miller et al. "ReadDepth: a parallel R package for detecting copy number alterations from short sequencing reads". In: *PloS one* 6.1 (2011). URL: https://github.com/chrisamiller/readdepth.

[142] Qian Liu et al. "Interrogating the "unsequenceable" genomic trinucleotide repeat disorders by long-read sequencing". In: *Genome medicine* 9.1 (2017), p. 65. URL: https://github.com/WGLab/RepeatHMM.

[143] Thomas M Keane, Kim Wong, and David J Adams. "RetroSeq: transposable element discovery from next-generation sequencing data". In: *Bioinformatics* 29.3 (2013), pp. 389–390. URL: https://github.com/tk2/RetroSeq.

[144] Tao Jiang et al. "rMETL: sensitive mobile element insertion detection with long read realignment". In: *Bioinformatics* 35.18 (2019), pp. 3484–3486. URL: https://github.com/hitbc/rMETL.

[145] Mitchell R Vollger et al. "Long-read sequence and assembly of segmental duplications". In: *Nature methods* 16.1 (2019), p. 88. URL: https://github.com/mrvollger/SDA.

[146] Mark JP Chaisson et al. "Resolving the complexity of the human genome using single-molecule sequencing". In: *Nature* 517.7536 (2015), p. 608. URL: https://github.com/EichlerLab/pacbio_variant_caller.

[147] John Huddleston et al. "Discovery and genotyping of structural variation from long-read haploid genome sequence data". In: *Genome research* 27.5 (2017), pp. 677–685. URL: https://github.com/EichlerLab/smrtsv2.

[148] Fritz J Sedlazeck et al. "Accurate detection of complex structural variations using single-molecule sequencing". In: *Nature methods* 15.6 (2018), p. 461. URL: https://github.com/fritzsedlazeck/Sniffles.

[149] Jan Schröder et al. "Socrates: identification of genomic rearrangements in tumour genomes by re-aligning soft clipped reads". In: *Bioinformatics* 30.8 (2014), pp. 1064–1072. URL: http://bioinf.wehi.edu.au/socrates.

[150] Steven N Hart et al. "SoftSearch: integration of multiple sequence features to identify breakpoints of structural variations". In: *PloS one* 8.12 (2013). URL: https://github.com/Steven-N-Hart/softsearch.

[151] Christoph Bartenhagen and Martin Dugas. "Robust and exact structural variation detection with paired-end and soft-clipped alignments: SoftSV compared with eight algorithms". In: *Briefings in bioinformatics* 17.1 (2016), pp. 51–62. URL: https://sourceforge.net/projects/softsv/.

[152] Junho Kim et al. “SoloDel: a probabilistic model for detecting low-frequent somatic deletions from unmatched sequencing data”. In: *Bioinformatics* 31.19 (2015), pp. 3105–3113. URL: http://sourceforge.net/projects/solodel/.

[153] Colby Chiang et al. “SpeedSeq: ultra-fast personal genome analysis and interpretation”. In: *Nature methods* 12.10 (2015), p. 966. URL: https://github.com/hall-lab/speedseq.

[154] Zhen Zhang et al. “Sprites: detection of deletions from sequencing data by re-aligning split reads”. In: *Bioinformatics* 32.12 (2016), pp. 1788–1796. URL: https://github.com/zhangzhen/sprites.

[155] Ramesh Rajaby and Wing-Kin Sung. “SurVIndel: improving CNV calling from high-throughput sequencing data through statistical testing”. In: *Bioinformatics* (2019). URL: https://github.com/Mesh89/SurVIndel.

[156] Danny Antaki, William M Brandler, and Jonathan Sebat. “SV2: accurate structural variation genotyping and de novo mutation detection from whole genomes”. In: *Bioinformatics* 34.10 (2018), pp. 1774–1777. URL: https://github.com/dantaki/SV2.

[157] Jeremiah A Wala et al. “SvABA: genome-wide detection of structural variants and indels by local assembly”. In: *Genome research* 28.4 (2018), pp. 581–591. URL: https://github.com/walaj/svaba.

[158] Bruno Zeitouni et al. “SVDetect: a tool to identify genomic structural variations from paired-end and mate-pair sequencing data”. In: *Bioinformatics* 26.15 (2010), pp. 1895–1896. URL: http://svdetect.sourceforge.net/.

[159] Xuefang Zhao et al. “Resolving complex structural genomic rearrangements using a randomized approach”. In: *Genome biology* 17.1 (2016), p. 126. URL: https://github.com/mills-lab/svelter.

[160] Rendong Yang et al. “Integrated analysis of whole-genome paired-end and mate-pair sequencing data for identifying genomic structural variations in multiple myeloma”. In: *Cancer informatics* 13 (2014), CIN–S13783. URL: https://github.com/cauyrd/SVfinder.

[161] David Heller and Martin Vingron. “SVIM: structural variant identification using mapped long reads”. In: *Bioinformatics* 35.17 (2019), pp. 2907–2915. URL: https://github.com/eldariont/svim.

[162] Kim Wong et al. “Enhanced structural variant and breakpoint detection using SVMerge by integration of multiple detection methods and local assembly”. In: *Genome biology* 11.12 (2010), R128. URL: http://svmerge.sourceforge.net.

[163] Jin Zhang, Jiayin Wang, and Yufeng Wu. “An improved approach for accurate and efficient calling of structural variations with low-coverage sequence data”. In: *BMC bioinformatics*. Vol. 13. S6. Springer. 2012, S6. URL: https://sourceforge.net/projects/svseq2/.

[164] Satomi Mitsuhashi et al. “Tandem-genotypes: robust detection of tandem repeat expansions from long DNA reads”. In: *Genome biology* 20.1 (2019), p. 58. URL: https://github.com/mcfrith/tandem-genotypes.

[165] Jiantao Wu et al. "Tangram: a comprehensive toolbox for mobile element insertion detection". In: *BMC genomics* 15.1 (2014), p. 795. URL: https://github.com/jiantao/Tangram.

[166] Eunjung Lee et al. "Landscape of somatic retrotransposition in human cancers". In: *Science* 337.6097 (2012), pp. 967–971. URL: http://compbio.med.harvard.edu/Tea/.

[167] Jiali Zhuang et al. "TEMP: a computational method for analyzing transposable element polymorphism in populations". In: *Nucleic acids research* 42.11 (2014), pp. 6826–6838. URL: https://github.com/JialiUMassWengLab/TEMP.

[168] Jesper Eisfeldt et al. "TIDDIT, an efficient and comprehensive structural variant caller for massive parallel sequencing data". In: *F1000Research* 6 (2017). URL: https://github.com/J35P312/TIDDIT.

[169] Guofeng Meng et al. "TSD: A computational tool to study the complex structural variants using PacBio targeted sequencing data". In: *G3: Genes, Genomes, Genetics* 9.5 (2019), pp. 1371–1376. URL: https://github.com/menggf/tsd.

[170] Alexandre Gillet-Markowska et al. "Ulysses: accurate detection of low-frequency structural variations in large insert-size sequencing libraries". In: *Bioinformatics* 31.6 (2015), pp. 801–808. URL: http://www.lcqb.upmc.fr/ulysses.

[171] Fatih Karaoglanoglu et al. "Characterization of segmental duplications and large inversions using Linked-Reads". In: *bioRxiv* (2018), p. 394528. URL: https://github.com/BilkentCompGen/valor.

[172] Fereydoun Hormozdiari et al. "Combinatorial algorithms for structural variation detection in high-throughput sequenced genomes". In: *Genome research* 19.7 (2009), pp. 1270–1278. URL: https://github.com/BilkentCompGen/tardis.

[173] Arda Soylev et al. "Toolkit for automated and rapid discovery of structural variants". In: *Methods* 129 (2017), pp. 3–7.

[174] Qingguo Wang, Peilin Jia, and Zhongming Zhao. "VERSE: a novel approach to detect virus integration in host genomes through reference genome customization". In: *Genome medicine* 7.1 (2015), p. 2. URL: https://bioinfo.uth.edu/VirusFinder/.

[175] Yunxin Chen et al. "VirusSeq: software to identify viruses and their integration sites using next-generation sequencing of human cancer tissue". In: *Bioinformatics* 29.2 (2013), pp. 266–267. URL: http://odin.mdacc.tmc.edu/~xsu1/VirusSeq.html.

[176] Zev N Kronenberg et al. "Wham: identifying structural variants of biological consequence". In: *PLoS computational biology* 11.12 (2015). URL: https://github.com/zeeev/wham.

[177] Menachem Fromer and Shaun M Purcell. "Using XHMM software to detect copy number variation in whole-exome sequencing data". In: *Current protocols in human genetics* 81.1 (2014), pp. 7–23. URL: https://bitbucket.org/statgen/xhmm.

[178] Li C Xia et al. "Identification of large rearrangements in cancer genomes with barcode linked reads". In: *Nucleic acids research* 46.4 (2017), e19–e19. URL: https://bitbucket.org/charade/zoomx.

[179] Le Zhang et al. "Comprehensively benchmarking applications for detecting copy number variation". In: *PLoS computational biology* 15.5 (2019), e1007069.

[180] Wanyu Bai Le Zhang, Na Yuan, and Zhenglin Du. "Correction: Comprehensively benchmarking applications for detecting copy number variation". In: *PLoS computational biology* 15.9 (2019).

[181] José Marcos Moreno-Cabrera et al. "Benchmark of tools for CNV detection from NGS panel data in a genetic diagnostics context". In: *BioRxiv* (2019), p. 850958.

[182] *Langmead Lab Teaching Materials @ John Hopkins University*. URL: http://www.langmead-lab.org/teaching-materials/ (visited on 01/27/2020).

[183] Saul B Needleman and Christian D Wunsch. "A general method applicable to the search for similarities in the amino acid sequence of two proteins". In: *Journal of molecular biology* 48.3 (1970), pp. 443–453.

[184] Martin Odersky and al. *An Overview of the Scala Programming Language*. Tech. rep. IC/2004/64. Lausanne, Switzerland: EPFL, 2004.

[185] Matei Zaharia et al. "Apache spark: a unified engine for big data processing". In: *Communications of the ACM* 59.11 (2016), pp. 56–65.

[186] Aleix Lafita et al. "BioJava 5: A community driven open-source bioinformatics library". In: *PLoS computational biology* 15.2 (2019), e1006791.

[187] Mathieu Bastian, Sebastien Heymann, and Mathieu Jacomy. "Gephi: an open source software for exploring and manipulating networks". In: *Third international AAAI conference on weblogs and social media*. 2009.

[188] John Ellson et al. "Graphviz—open source graph drawing tools". In: *International Symposium on Graph Drawing*. Springer. 2001, pp. 483–484.

[189] Peter Deutsch et al. *GZIP file format specification version 4.3*. Tech. rep. RFC 1952, May, 1996.

[190] Shunichi Kosugi et al. *Comprehensive evaluation of structural variation detection algorithms for whole genome sequencing. Data set and source code*. 2019. URL: https://github.com/stat-lab/EvalSVcallers.

[191] Justin M Zook et al. "A robust benchmark for germline structural variant detection". In: *BioRxiv* (2019), p. 664623.

[192] Weichun Huang et al. "ART: a next-generation sequencing read simulator". In: *Bioinformatics* 28.4 (2012), pp. 593–594.

[193] Brent S Pedersen and Aaron R Quinlan. "Mosdepth: quick coverage calculation for genomes and exomes". In: *Bioinformatics* 34.5 (Oct. 2017), pp. 867–868. ISSN: 1367-4803.

[194] *bigWig Track Format*. URL: https://genome.ucsc.edu/goldenPath/help/bigWig.html (visited on 10/23/2019).