UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Università degli Studi di Padova
Dipartimento di Biologia, CRIBI

SCUOLA DI DOTTORATO DI RICERCA IN BIOSCIENZE E BIOTECNOLOGIE
INDIRIZZO BIOTECNOLOGIE, CICLO XXIV

Identification of Structural Variations in Resequenced Genomes using Paired-End or Mate-Pair Sequences

Direttore della Scuola: *Ch.mo Prof Giuseppe Zanotti*
Coordinatore d'Indirizzo: *Ch.mo Prof Giorgio Valle*
Supervisore: *Ch.mo Prof Giorgio Valle*

Dottorando: *Gianpiero Zamperin*

A. A. 2011-2012

ABSTRACT

Next Generation Sequencing (NGS) allows the production of a lot of data in cheaper ways than the traditional Sanger technology. The huge amount of data that recently has been obtained with NGS resulted in a fast production of the draft sequence of many genomes, both from eukaryotic and prokaryotic organisms. The Human Genome Project was completed in 2003 ([1]), less than 10 years ago: it cost billions of dollars and involved dozens of laboratories from all around the world. Currently, NGS allows to get the equivalent of a human genome in few weeks, at a price of 10,000 dollars. This amazing increase of performance has opened new possibilities in the biological field: for example now it is possible to genetically compare entire organisms, analyse ancient DNA, study genetic diseases at a level that was unbelievable until few years ago. Some of the main fields that can be improved with this technology are: genomics (for example, genome assembly and structural variations detection), transcriptomics (for example, analysis of gene expression, gene prediction and alternative splicing) and epigenetics.

The huge amount of data that has been produced needs to be analysed; it is very unlikely that such analysis could be done manually, so new bioinformatic methods are needed to speed up the process. There is a need for optimizing computational resources to efficiently store NGS data, but also the need for new algorithms, specifically designed for NGS data, for instance to overcome one of the major limitations of new sequencing technologies: the short length of the individual sequences (generally called 'reads') that can be delivered by NGS machines. From one side, NGS can produce several hundred times more reads than traditional sequencing, but on the other hand these reads are much shorter: about 50-100 bases instead than 500-1000 bases of Sanger sequencing. This makes the analysis of the data more difficult , in particular for genomic repeats that can be resolved only with longer reads.

Currently the NGS machines that are mostly used are Solexa (Illumina),

454 (Roche) and SOLiD (Applied BioSystems). The first one uses a method similar to Sanger sequencing, while the other two use different technologies, respectively pyrosequencing and sequencing-by-ligation. The length of the reads is variable: 454 produces reads of about 400 bases, while the other two produce reads of length between 35 and 100 bases. The three platforms differ also in their throughput that continuously improves over time; currently the 454 produces about one million reads per run, while Solexa and SOLiD can produce several hundred millions reads per run.

These platforms can be used to sequence different types of libraries, including paired-end and mate-pair libraries. They are libraries that allow the sequencing of the ends of DNA fragments; as a result, pairs of sequences are produced, that must map at a distance compatible with the length of the library fragments. When used for re-sequencing individual genomes, these libraries generate a lot of links ('arcs'), one for each pair of mapped reads, that must be compatible with the length of the library fragments. The main objective of my PhD thesis is to prove that it should be possible to identify with high accuracy any structural variation occurring in individual genomes, using the data from paired-end and mate-pair libraries. The accuracy of this analysis should improve with the density of arcs that are covering the genome; therefore, the large number of arcs that can be generated by NGS platforms offers a great opportunity for structural variation studies.

Structural variations are an aspect of the genome whose importance has become evident only in the past few years: before, even their existence was doubtful. It has been recently observed that in adult genomes hundreds of structural variations are present, which may be associated with cancer of other diseases (for example Parkinson's disease). Several tools have been developed to detect structural variations, based on comparative genome hybridization and, more recently, on NGS. In the latter case, the tools available are still far from being able to exploit the full potential of the NGS data, both in terms of sensitivity and specificity. The aim of my PhD was to investigate this problem and to create a bioinformatic tool able to detect structural variations with high accuracy. At the beginning I focused only on SOLiD data, then I extended my analysis also to Solexa data (and, virtually, 454). As a final result I created SV_finder, a program able to work both in base and color space. As an input it requires the list of paired-end or mate-pair reads mapped on a known reference genome; the output is a list of structural variations found on the basis of data and parameters used.

SOMMARIO

Le nuove tecnologie di sequenziamento (NGS) consentono di ottenere moltissimi dati a costi contenuti rispetto al tradizionale sequenziamento Sanger. L'enorme mole di dati che recentemente è stata prodotta con le NGS ha portato ad una veloce produzione di bozze di sequenza di molti genomi, sia eucariotici sia procariotici. Il Progetto Genoma Umano fu completato nel 2003 ([1]), meno di 10 anni fa: costò miliardi di dollari e interessò decine di laboratori in tutto il mondo. Attualmente le NGS consentono di produrre l'equivalente di un genoma umano in poche settimane, al costo di 10000 dollari. L'incredibile aumento di prestazione ha aperto nuove possibilità in campo biologico: ad esempio ora è possibile comparare geneticamente interi organismi, analizzare DNA antico, studiare malattie genetiche ad un livello ritenuto incredibile fino a pochi anni fa. Alcuni dei principali campi che possono essere migliorati con questa tecnologia sono: genomico (ad esempio assemblaggio di genomi e identificazione di variazioni strutturali), trascrittomico (ad esempio predizione genica e splicing alternativi) ed epigenetico.

L'enorme mole di data che è stata prodotta deve essere analizzata; è molto improbabile che tale analisi sia fatta manualmente, quindi nuovi metodi bioinformatici sono richiesti per velocizzare il processo. C'è il bisogno di ottimizzare le risorse computazionali per memorizzare efficaciemente i dati NGS, ma anche il bisogno per nuovi algoritmi, concepiti specificatamente per i dati NGS, ad esempio per superare una delle maggiori limitazioni delle nuove tecnologie di sequenziamento: la corta lunghezza delle singole sequenze (in generale chiamate 'reads') che può essere prodotta dalle macchine NGS. Da una parte, le NGS possono produrre centinaia di volte più reads del sequenziamento tradizionale, ma dall'altra parte queste reads sono molto più corte: circa 50-100 basi invece che 500-1000 basi del sequenziamento Sanger. Ciò rende più difficoltosa l'analisi dei dati, particolarmente per le repeat genomiche che possono essere risolte solo con read più lunghe.

Attualmente le macchine NGS più utilizzate sono il Solexa (Illumina), il 454 (Roche) e il SOLiD (Applied BioSystems). Il primo usa un metodo simile al sequenziamento Sanger, mentre gli altri due usano tecnologie differenti, rispettivamente pyrosequencing e sequencing-by-ligation. La lunghezza delle read è variabile: il 454 produce read di circa 400 basi, mentre gli altri due producono read di lunghezza compresa tra 35 e 100 base. Le tre piattaforme differiscono anche nel rendimento che continuamente migliora nel tempo: attualmente il 454 produce circa un milione di read per corsa, mentre Solexa e SOLiD possono produrre molte centinaia di milioni di read per corsa.

Queste piattaforme possono essere usate per sequenziare differenti tipi di librerie, incluse le librerie paired-end e mate-pair. Esse sono librerie che permettono di sequenziare le estremità di una frammento di DNA; come risultato vengono prodotte paia di sequence che devono mappare ad un distanza compatibile con la lunghezza dei frammenti della libreria. Quando usate per ri-sequenziare genomi singoli, queste librerie generano molti link ('archi'), uno per ogni coppia di read mappate, che devono essere compatibili con la lunghezza dei frammenti della libreria. L'obiettivo principale della mia tesi di dottorato è dimostrare che dovrebbe essere possibile identificare con elevata accuratezza qualsiasi variazione strutturale che si presenti nei genomi di singole persone usando i dati di librerie paired-end e mate-pair. L'accuratezza di questa analisi dovrebbe migliorare con la densità di archi che coprono il genoma; quindi, il grande numero di archi che può essere generato dalle piattaforme NGS offre una grande opportunità per gli studi su variazioni strutturali.

Le variazioni strutturali sono un aspetto del genoma la cui importanza è diventata evidente solo negli ultimi anni; prima, perfino la loro esistenza era messa in dubbio. Recentemente si è osservato che in genomi adulti sono presenti centinaia di variazioni strutturali che possono essere associate a cancro o altre malattie (per esempio il morbo di Parkinson). Molti strumenti sono stati sviluppato per identificare le variazioni strutturali, basati sulla comparative genome hybridization e, più di recente, sulle NGS. Nell'ultimo caso, gli strumenti disponibili sono molto lontani dall'essere capaci di sfruttare il pieno potenziale dei dati NGS, sia in termini di sensibilità che specificità. Scopo del mio dottorato è esaminare questo problema e creare uno strumento bioinformatico capace di identificare le variazioni strutturali con elevata accuratezza. Inizialmente mi sono concentrato solo sui dati SOLiD, in seguito ho esteso la mia analisi anche ai dati Solexa (e, potenzialmente, 454). Come risultato finale ho ideato SV_finder, un programma capace di funzionare sia

in base che color space. Come input richiede una lista delle read paired-end o mate-pair mappate su un genome conosciuto di riferimento; l'output è una lista di variazioni strutturali trovate in base ai dati e parametri usati.

Contents

List of Figures

Chapter 1

Introduction

1.1 Aim

The aim of my PhD research was the development and application of bioinformatic tools for the identification of differences between an individual human genome and the reference human genome. More specifically, the individual human genome belonged to a patient affected by a blood coagulation disease. It was known that the blood coagulation disease had a genetic nature, but the affected gene was unknown and standard techniques such as linkage analysis had not been successful. The general strategy that we wanted to apply was:

- to sequence the genome using mate pair libraries
- to find the differences with the reference human genome, mainly SNPs and structural variations
- to order the differences from the most to the less likely to cause the disease

Immediately, at the beginning of the project, it was clear that the work to do was massive, but also very innovative. The technology to produce the reads was still uncertain, especially for mate pair libraries. Although the mate pair technology had been described as a potential sequencing strategy, the bioinformatic tools available for studying structural variations using mate pairs were not available. Therefore I decided to focus only on structural variations and on the possibility to develop a suitable bioinformatic tool to perform

mate pair analysis. In conclusion, the aim of my PhD was the creation of a bioinformatic tool able to detect structural variations in a complex genome such as the human genome, using paired-end or mate-pair libraries.

1.2 Structural Variations: General Overview

Structural variations typically affect genomic sequences from $1Kb$ to $3Mb$: they are much larger than SNP, but smaller than chromosome abnormalities visible at a macroscopic level. Structural variations can be of many types:

deletion : a region of DNA is missing

insertion : in a certain point of the DNA, a sequence of DNA is inserted

inversion : a region of DNA is present at the same point but with opposite orientation

translocation : a region of DNA is missing from its normal position and re-inserted at another point of the genome

duplication : a region of DNA is duplicated

Structural variations are often associated with genetic diseases, although most of them are not. Recently, evidence showed that individual genomes contain hundreds of structural variations, and so they probably are an important contribution to diversity. Also it is likely that they have been an important force in shaping the genome during evolution. Other structural variations may arise in somatic cells and may be the cause of cancer.

1.2.1 Definition

Structural variation can be divided into two main categories: copy-number variants (CNV) and copy-count invariant[12]. The terminology refers to a change or not in the total content of DNA: for example, an inversion changes the order of nucleotides in the DNA sequence, but not the total number of nucleotides, so it is a copy-count invariant; on the other hand, a deletion erases a certain number of nucleotides from the DNA sequence, so it is a copy-number variant. In general, a structural variation involves sequences larger than $1Kb$, but this is not an absolute limit, because often under the definition

of 'structural variation' sometimes are included changes of sequences shorter than that $1Kb$[1].

Deletion

A deletion is a CNV. Probably it is the most common among the structural variations, due to the easiness of its creation: when DNA breaks in two near[2] points and the repair happens without the central region, resulting in a deletion :

$$AAACGACTTGTTGT \mid tagcga \mid CACACGTCTACGCT$$

$$1|2 \qquad 3|4$$

if the repair mechanism fails and joins 1 with 4, the sequence *tagcga* between 1 and 4 will become a deletion. A deletion can also be caused by errors in chromosomal crossover during meiosis. When a deletion occurs, some genetic material is lost: if the deletion interests a coding region, the protein coded can lack one or more domains, resulting in a partial or complete loss of function, or there could be an alteration of the coding frame, resulting in a frameshift mutation. If the deletion occurs in a regulatory region, the consequences are obvious: a regulatory control is lost, so this could have very severe effects, depending on what that regulatory region controls.

Insertion

An insertion is a CNV. It can happen due to unequal crossover during meiosis, but also from from a break of the DNA in a certain point and joining of the break ends with a foreign DNA sequence:

$$AAACGACTTGTTGT \mid CACACGTCTACGCT$$

$$1|2$$

[1] in this PhD thesis, for 'structural variation' is taken as something that involves changes from dozen of bases to many thousands

[2] 'near' means 'at a reasonable distance': two breaks ten bases apart are likely to be repaired, whereas two breaks separated by one million bases are very unlikely to be repaired

DNA breaks in the pipe (|) point; now 1 2 are rejoined with a foreign sequence ($tagcga$) inside

$$AAACGACTTGTTGT\ |tagcga|\ CACACGTCTACGCT$$

$$1| \qquad 2$$

The foreign sequence can be a sequence not present in the genome or a sequence that is present elsewhere in the genome. An insertion cause a frameshift mutation if it happens in a coding region and the number of nucleotides inserted is not a multiple of three; in the latter case, with inserted sequences multiples of three, the resulting protein will have more amino acids, and this may or may not have severe consequences on its function. An insertion in a regulatory region could modify or completely cut away the coded regulation.

Inversion

An inversion is a copy-count invariant. It happens when DNA breaks in two near points (the same situation as deletion) and the ends are rejoined as follows:

$$AAACGACTTGTTGT\ |\ tagcga\ |\ CACACGTCTACGCT$$

$$1|2 \qquad 3|4$$

1 with 3 and 2 with 4. An inversion does not involve a loss of genetic information, but simply rearranges the linear gene sequence, so it can have severe consequences or nothing at all. If an entire coding region is affected by an inversion, probably nothing would happen: the coding sequence would remain the same, and so would the amino acids of the encoded protein. Maybe the regulation could be affected, depending on the position of the inversion. If an inversion interests the central portion of the coding region, the coded protein can have a change of frame and the amino acid sequence could completely change, resulting in a very severe modification.

Translocation

A translocation is a copy-count invariant: chromosomal segment that changes position within a genome without change in the total DNA content[13]. Basically a translocation is a deletion and a paired insertion, because the starting

position of the translocated segment is a deletion and the end position is an insertion.

Duplication

A duplication is a CNV; it is the opposite of a deletion: an unequal crossing-over that occurs during meiosis between misaligned homologous chromosomes that can produce a duplication. It depends on the number of repetitive elements between the two chromosomes: the more repetitive elements are present, the easier it is to misalign the chromosomes.

If a duplication occurs in a coding region, a second copy of that gene is created. This copy is often not under selective pressure, so it accumulates mutations faster than the original copy of the gene without negative consequences.

When a duplication is located in a repetitive region due to the nature of paired-end and mate-pair library (section 1.4.2), reads that cover a duplicated region align with good match in many places, so it is impossible to recognize the 'right' mapping position. When reads are paired[3], a lot of combinations are found, some with right distances and some not: how can one know if the combinations with wrong distances are sequencing errors or true duplications? It is very difficult to answer. Some suggestions involving the analysis of mate pairs have been proposed [2], but bioinformatic tools have yet to be developed; as a result, duplications are never found.

1.2.2 Brief Story

About 20 years ago, in 1991, the Charcot-Marie Tooth (CMT) disease was the first discovered autosomal dominant disease due to a structural variation, a tandem duplication that leads to a gene overdosage effect (three copies of the normal gene). Currently it is widely accepted that a lot of genetic disorders are caused by (or have a relation with) structural variations, for example Williams-Beuren and Prader-Willi syndromes, autism[14] and Parkinson's disease.

Structural variations are not only linked to disease phenotypes. Sometimes they interest genes involved in environmental response, immunity and olfactory receptors, or are responsible for a disease susceptibility. Initially

[3] i.e. for every read its sibling is found

SNPs were thought to be the major contributor to genomic variation, but it is more and more evident (thanks to the development of new technologies such as high-throughput sequencing) that structural variations give a significant contribution to genome variation, evolutionary shaping and genetic disease. However it is still a poorly understood aspect of the genome, so further studies are needed for more robust conclusions.

1.2.3 Importance

The interest in structural variations arose when it became clear that they are related to genetic disease. Previously the same thing happened to SNPS. It was thought that SNPs were the main form of genetic variations, therefore a person would have a certain phenotype due to the SNPs that she/he has. It was reasonable to think that knowing the SNPs meant potentially knowing how to 'shape' a human being. This was a very simple assumption, but the main idea still holds: differences between people are due to differences between their genomes, but structural variations also contribute to the diversity. Finding SNPs is relatively easy: from the alignment of a sequence against the reference, all mismatches could be SNPs. However with structural variations things are more difficult and until the availability of NGS (section 1.3.2) the evidence was very limited.

Currently we have machines and methods to find structural variations, and a lot of data indicate that they are often related to genetic diseases and play a large role in genomic variations. Now the possibility of linking phenotypes to differences between genomes is closer: with NGS data both SNPs and structural variations can be found and differences between genomes can be identified. The main problem is to link those differences with phenotypes, but this was also the main problem that arose when the sequencing era began in 1977. In those years, scientists thought that after knowing the DNA sequence of an organism everything could be understood. Maybe it is true, but now we realize that having the DNA sequence is like having the pieces of an immense puzzle and that most of the work is not the DNA sequencing but putting together all the pieces.

Currently, it has become evident that structural variations are related to diseases, resulting in a lot of interest in them. The first thing to do in order to cure a genetic disease is to know the involved gene and the defect it carries; if this defect is due to a structural variation or a SNP is very relevant both in terms of diagnosis and possible cures.

1.2.4 How to Find Them

Before NGS, Hybridization-based microarray approaches were the common way to find structural variations. These approaches are mainly based on array comparative genomic hybridization (array CGH) and SNP microarrays. Both technologies find copy number gains or losses compared to a reference, with some important differences.

A CGH array is a platform with a set of hybridization targets; two labelled samples, test and reference, are hydridized and the signal ratio is used to proxy copy number. A problem is that, with only one reference sample, it is impossible to distinguish between a loss in the reference and a gain in the test, so the reference is very important in the test. In the beginning, with the CGH array it was possible to discover only structural variation longer than $10Kb$, anyway this highlighted a great amount of CNVs[4] in healthy individuals; more recently, technology was improved and, for research purposes, now the resolution allows the discovery of CNVs down to $500bp$, using approximately 42 million probes spread across twenty 2.1 million probe arrays. CGH arrays are used in clinical diagnostics.

SNP arrays are platforms where only one samples per microarray is hybridized, and so the ratio is computed using intensities measured at each probe across many samples. Probes are allele specific, so the CNV sensitivity is increased, but the signal-to-noise ratio offered per probe is often lower than the best CGH array could offer. For these reasons, SNP arrays were most used as complements to array CGH for fine-mapping regions.

With the advent of NGS technologies, studies of structural variations were vastly improved, but they proved to be a new challenge, because NGS data needed new algorithms to exploit all available data. The general strategy focuses on mapping the reads to a reference and then identifying those signatures different from the expected ones in order to detect structural variations. Four main methods were developed.

Read-pair methods use mate reads that must align with a certain distance and orientation; in principle, a structural variation changes one or both of those feature of a mate pair (distance and/or orientation), so it is possible to detect all classes of variation. The idea is simple: mate reads with something 'wrong' are clustered together; a cluster identifies a structural variation; with some sort of refinement, breakpoints are found.

[4]only structural variation with a gain or loss of genetic material can be found with Hybridization-based microarray approaches

Read-depth methods assume that sequences align on the reference genome with a certain distribution (typically a Poisson); when the distribution shows a divergence, it is investigated to discover duplications and deletions: a duplication should show a higher coverage, while a deletion should show a reduced read depth. Breakpoint detection is a problem of this approach.

Split-read methods use a splice-alignment to detect both structural variations and breakpoints at the same time. The idea is to align sequences allowing the split of the sequence itself and detect the split read signature. Breakpoints are found with a base precision, but currently this approach is very limited by the short sequence of NGS reads.

Sequence assembly methods is the approach among the four possible that is still in its infancy. The idea is simple: sample genome is *de novo* assembled and it is compared to the reference genome to find all the possible structural variations, without limitations in type, length, or number of copies. In practice, this idea is not feasible, because the assembly requires long sequences, and NGS produces short sequences. Some approaches are still in development, for example a combination of *de novo* and local-assembly algorithms to create a raw assembly (only contigs) that are compared to the reference, but we are still very far from genome assembly using NGS sequences.

1.2.5 Old vs New Methods

Microarrays has some severe limitations:

- can only discover CNV
- provide no information on the location of duplicated copies
- in general cannot resolve breakpoints with a base precision
- have a reduced sensitivity in the detection of single copy gains
- assume that each location is diploid is the reference genome, but this is not always true, especially in the duplicated and repeat-rich regions

The main advantage they have is the low cost compared to NGS: whenever a screening of a large set is needed[5], NGS costs too much because, despite

[5]for example when a screening of hundreds of individual is done in order to detect a rare disease related to a structural variation

the decreasing cost, sequencing hundreds of different genomes requires a lot of money, while microarrays are a lot cheaper.

Methods using NGS data have the main advantage of being able to detect most types of structural variations in a single experiment, although each one of the four approaches has limitations: for example, read-depth methods are the only ones able to predict absolutely copy number, but they poorly detect breakpoints. With the decreasing cost of sequencing and improving of algorithms, NGS methods will probably become the most used approach.

1.3 Sequencing

Sequencing is the way to know the types and order of bases that form a piece of DNA (and, in general, of an acid nucleic). First sequencing technique was developed by Frederick Sanger and Alan R. Coulson in the 1977[11]; in the following years it was improved to reduce sequencing costs and required time. Knowing the DNA sequence is the most important and basic information in many biology fields: DNA sequence codes for protein sequence and location, from which the protein structure (or lack of) depends[6], from which protein function depends. Life is based on protein functions, so knowing DNA sequence is a way to know life. As everyone knows, in practice it is not so simple and linear: apart from proteins, also RNA plays an important role in life; moreover, DNA carries information in different ways than sequence, for example methylation level. Still DNA sequencing play a fundamental role in most of current biology fields.

1.3.1 Sanger Sequencing

Sanger Sequencing is based on the fact that a nucleotide lacking hydrophilic groups on 2^i and 3^i carbons (ddNTP) cannot bind to another nucleotide. The idea of sequencing is simple as well as brilliant:

- a single-strand DNA is used as template to generate the complementary strand
- in the mixture used for the reaction are present one type of ddNTP; four mixtures are used, each one with a different ddNTP

[6]protein structure depends on the interaction between amino acids and environment, but the latter depends on where the protein is located, its location precisely

- four reactions take place: in each reaction, whenever a ddNTP is incorporated into the reaction, elongation is stopped
- after the four reactions, a collection of fragments truncated at a certain position is obtained; these fragments differ in length
- with a gel electrophoresis, fragments are separated from the shortest to the longest
- if the ddNTP are marked[7], the sequence of the template can be determined by evaluating the position of fragments in the gel

Sanger Sequencing was used in the Human Genome Project, a sequencing milestone showing how massive could be a sequencing project and the importance of developing new and better sequencing technologies.

1.3.2 Next Generation Sequencing

At the end of 1990s, there was a common idea of needing an improvement in sequencing technologies. NGS were developed as an answer to this sense. One major drawback about Sanger Sequencing was the time and effort needed to sequence DNA: long DNA sequences were fragmented, then each fragment was cloned into a vector and amplified with *E. coli*; finally DNA was purified, sequenced with Sanger's and assembled into the original long sequence. It was a lot of work to do. NGS was developed to avoid all this work (or, at least, dramatically decrease costs, time and people required) and maintain other features such as quality and sequence length. Unfortunately, only the reduction of time and cost was successful, because NGS reads are shorter that Sanger's reads and less accurate, but they come in much greater numbers, so low quality and short length could be partially solved with mere high coverage. Many NGS machines were developed, but currently the most used are Illumina, 454 and SOLiD.

Roche (454) sequencer

The 454 was the first NGS to make its appearance; it is based on pyrosequencing:

[7] in the beginning they were marked with radioactivity, then fluorescent labels were used

- luciferase reactions are used to produce light
- when a nucleotide is added to the new nucleotide chain in the elongation process, a pyrophosphate (PPi) is released
- PPi is converted to ATP by the ATP sulfurylase
- luciferase uses the ATP to produce light
- so, when the nucleotide chain is elongated, light is produced
- because the nucleotide added is known, the sequence can be determined:
  - 'A' nucleotide added $\longrightarrow$ no light
  - 'C' nucleotide added $\longrightarrow$ no light
  - 'G' nucleotide added $\longrightarrow$ no light
  - 'T' nucleotide added $\longrightarrow$ light, which means that the sequence has a 'T' in that position
  - repeat previous step for next positions

Sequences produced with 454 have great length (400 bp) and quality, not at the levels of Sanger's ones but greater than Illumina and SOLiD. The major drawback is homopolymers: the light signal is directly proportional to the number of base in the homopolymer for sequences of maximum 8 base length; for longer homopolymers, saturation occurs, so the signal is not reliable. Another drawback is the relative short length of sequences (although it is a common drawback in NGS), caused by the fact that in some cycles a small fraction of templates lose the right timing, leading to an emission of light in the wrong moment: this causes a background noise that increases with time, so sequences have a maximum length, after which the background noise is so high that the sequence is not at all reliable.

Illumina Genome Analyzer

Illumina uses the same chemistry of Sanger Sequencing, but the amplification is made *in vitro*:

- DNA fragment ends are repaired and two adaptors are linked

- denatured DNA are linked to a flow cell, in which primers complementary to the adaptors are present
- a PCR in solid phase is done (bridge PCR) in repeated cycle to amplify all DNA fragments
- thanks to the bridge PCR, the flow cell has different clusters each one formed with the same DNA molecule
- Sanger sequencing is made:
  - a mixture with all four nucleotides labelled with a fluorescent dye is added; these nucleotides cannot further elongate the chain, unless the fluorescent dye is cut away
  - elongation takes place
  - nucleotides not linked are washed away
  - a laser reads the fluorescence in each cluster (fluorescence emitted depends on the last nucleotides labelled!)
  - fluorescent dyes are cut away and the entire process repeated

In this way, in each step a new nucleotide is read to sequence the entire DNA fragment in each cluster. Illumina reads are shorter (100 bp) than 454's reads, but the output is greater. The major drawback is substitutions, due to the simultaneous addition of all four nucleotides: Illumina reads often show errors on nucleotide type, rather that deletions or insertions of one or more bases.

Applied Biosystem SOLID^{TM} sequencer

SOLiD uses a peculiar chemistry: it is based on a ligation to sequence a DNA fragment:

- oligonucleotides of 8 bases are used
- an oligonucletide has the first 3 bases degenerated (all the possible combinations of 3 nuleotides are present without distinction), the last 3 are universal nucleotides (they link with every nucleotide indiscriminately), and the two central bases are the ones used to determine the sequence (the 16 possible combinations are labelled with 4 different fluorescent dyes)

- there is a total of 1024 different nucleotides
- a mixture of all oligonucleotides is added
- oligonucleotides not linked are washed away
- fluorescence is read: di-base present in that location gives the color
- the previous steps are repeated for every position in the array to form a sequence of colors

If the first base (or a base in any position) is known, the sequence of colors has a unique meaning. On a theoretical level, an oligonucleotide of 8 bases link to the template, so the pairing is more specific than having a single nucleotide (as happens for 454, Illumina and even Sanger's), but despite that SOLiD has the highest rate of errors among NGS. Also the read length is short: from 25 to 50 bp. Moreover, sequences are in color space, which need specific bioinformatic tools to be analysed. These are several drawbacks, but they are outweighed by the high amount of data produced (the highest among NGS) and the use of color space to recognise errors and SNP. To sum up: a lot of more data with less quality and more problems.

1.3.3 Sanger's vs NGS

Next Generation Sequencing has opened a lot of new fields in biology, because it has made available more data than could be expected with Sanger sequencing. However, this does not mean that the same data is available: NGS is different from Sanger. There are some advantages, but also some drawbacks:

- Sanger sequencing needs a step of clone amplification in *E. coli* that is the major reason for the time and money required
- NGS does this step *in vitro*, with a great saving in both money and time
- Sanger sequencing can not be efficiently parallelized therefore only one sequence at a time can be produced.
- NGS efficiently parallelizes sequencing with no human effort

- Sanger sequencing reactions need great[8]
- NGS uses very small quantities of reagent volume, thus reducing the costs
- Sanger sequencing produces reads of thousands of bases
- NGS produces reads of dozens to hundreds of bases (different NGS technologies produces sequences of different length within that range)
- Sanger reads have high quality
- NGS reads have poorer quality that Sanger's

To sum up: Sanger sequencing produces less, longer and more correct reads at an increased cost and in more time, while NGS produces more, shorter and less correct reads at much less cost and less time. NGS has some disadvantages that could be solved with more coverage and new bioinformatic tools (for example, short reads cannot be used to create a new genome by overlapping them with tools that use Sanger reads, because in this task read length is critical).

In addition to Sanger sequencing, NGS machines differ each others. Each of them has certain features that make it more or less suitable for a research project, so it is important to choose the right sequencing method for the task undertaken. For example, for small-scale projects, Sanger sequencing is still the better technology to choose, while for large-scale projects, NGS is a must for its low cost and high amount of produced data; the question is: which NGS machine should one use? SOLiD seems very good in SNP detection, while 454's long reads (400 bp) are very suitable for genome assembly. NGS chosen depends on the projects, so currently no NHS machine is better than the others. Sometimes, the best approach is a mixed one, using data from more that one sequencing methods. It really depends from the project.

1.4 Libraries

When the DNA must be sequenced, it is not enough to simply extract the DNA from the wanted organism, put it into the sequencing machine and

[8] 'great' for molecular biology standard...in reality, microliter to milliliters of volume

await the results. DNA must be prepared and adapted to the use one wants. This means library preparation, in very simple steps:

1. wanted genetic material are extracted for the organism to be sequenced; it could be DNA, but also RNA (in the latter, RNA are retrotranscribed into cDNA)

2. DNA is fragmented into pieces with a certain process (it could be chemical or physical)

3. library is prepared from DNA pieces after fragmentation and is finally ready for the sequencing process

The experimental details about those 3 three steps are not important for the purpose of the present PhD thesis. The only important thing is the third step, i.e. the types of library that could be produced.

1.4.1 Fragment

Basically when DNA is fragmented and the entire fragment is sequenced, a fragment library is build. A sequenced fragment library gives a lot of sequences of good length that can be used to create a genome: whenever two sequences overlap, they are 'linked' to form a unique sequence. By the overlapping of many sequences, theoretically the entire sequenced genome can be build. Two main techniques uses fragment libraries: chromosome walking and whole genome shotgun sequencing. In chromosome walking three basic step are followed:

- a primer that matches the beginning of the DNA to be sequenced is used to synthesize a short DNA strand adjacent to the unknown sequence, starting with the primer

- the new short DNA strand is sequenced using the chain termination method

- the end of the sequenced strand is used as a primer for the next part of the long DNA sequence

So the sequencing of a long DNA region proceeds with small steps over the region itself.

In whole genome shotgun sequencing, DNA is fragmented into small pieces, which are sequenced to obtain reads. Multiple overlapping reads for the fragment DNA are needed. Using the overlapping ends of different reads, they are assembled into a continuous sequence.

1.4.2 Paired-End and Mate-Pair

When the DNA is fragmented, only fragments of certain length are selected and only their ends are sequenced, a paired-end or mate-pair library is build. Such library is made up of short sequences that are mated, so every sequence has a sibling (a 'mate'). Moreover, due to fragment selection based on length, two sibling sequences must have a certain distance, that depends on the fragment length. The experimental techniques used to obtain these libraries are different, but basically:

- only ends of the fragment are sequenced
- only one strand of the fragment is sequenced
- in general, in case of paired-end library, ends are sequenced in different directions, so if a read aligns against the reference, its sibling aligns with is reverse-complement
- in general, in case of a mate-pair library ends are sequenced in the same direction, so two sibling reads align against the reference on the same strand

Paired-end and mate-pair libraries are very suitable for structural variation detection, because, whenever the structure in DNA changes, also the way two sibling reads align changes: by analysing the wrong alignment, it is possible to detect structural variations. Also the sequenced tags are short, so the NGS machines, that produce shorter and more sequences than Sanger, are good candidates for paired-end and mate-pair library sequencing.

1.5 Distance Distribution

In a paired-end or mate-pair library, all reads have a sibling (also called 'mate') read and the two must be within a certain distance, as a result of the way the library is created: two sibling reads are the ends of the same DNA

fragment and their distance reflects the length of this fragment. During the library preparation, the size of the fragments is known and theoretically well defined, but:

- rather than a unique value, a range of values for distances between all pairs of reads is obtained and they should form a Gaussian distribution
- a lot of experimental variables can influence the fragment size, so in practice the distance distribution is computed after the sequencing and alignment of reads, using only reads well aligned[9]

The Gaussian distribution of distances between all sibling reads is called 'distance distribution' or simply 'distribution'. A Gaussian distribution has two very important values that define it: a mean value and a standard deviation; it has a central region, where most of the data falls, and two symmetrical tails where few data fall:

mean value : the mean value is the average value of all values forming the distribution; it is computed as the arithmetic mean A, with $A = \frac{1}{n}\sum_{i=1}^{n} a_i$. Sometimes other means are used, such as the weighted mean W, with $W = \frac{\sum_{i=1}^{n} w_i x_i}{\sum_{i=1}^{n} w_i}$. I always used A ($AvgLen$) and W (al_e)

standard deviation : the standard deviation is a measure of how much the values are scattered; the higher the standard deviation, the great the range size of values. A good Gaussian distribution has a low standard deviation. Moreover, it is known that a fixed percentage of values falls within a fixed number of standard deviations (figure 1.1).whatever the shape of distribution.

1.5.1 Problems

Sibling read distance should form a Gaussian distribution, but often, due to problems occurring in the library preparation, the distribution shape is different. For example, when the gel is cut to select fragments of only a certain size, some fragments of different lengths remain in the gel, producing an

[9]it is a recursive process: from library preparation, a certain value of fragment size is known; using that value, a first raw distance distribution is computed; with a second analysis of paired-end, the first distribution is refined

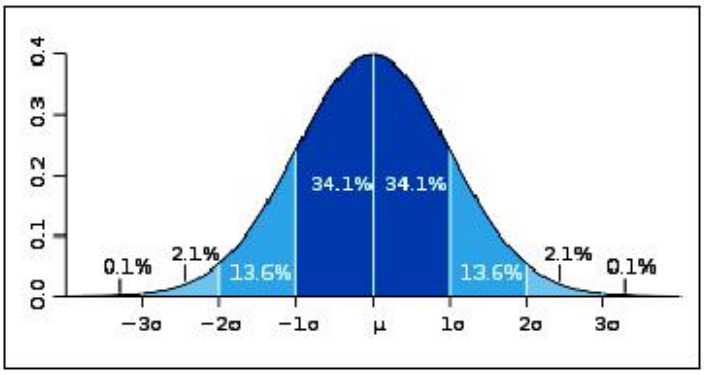

Figure 1.1: In a Gaussian distribution, 68.2% of values lies within one standard deviation (σ) from the average length (μ), 95.4% within two standard deviations and 99.6% within three standard deviations.

asymmetrical distribution, unbalanced to the left or to the right, depending from the length of fragments remaining, shorter or longer respectively. In the worst case, the 'foreign' fragments have a length so different from the wanted one that they form another peak: on the Gaussian distribution, two peaks appear, corresponding to the wanted fragments and those that contaminated the library.

1.6 State of the Art

With the appearance of structural variations as an important aspect of the genome, a lot of tools for their detection were developed. The idea is simple: whenever a structural variation affects a certain region of the genome, all the arcs from that region are changed, in length and/or orientation; so from the arcs that show something strange it is possible to detect the structural variations responsible for the strangeness.

This is only partially true: in practice, there are many reasons behind the strangeness of an arc:

1. preferential point of fragmentation in the DNA
2. preferential point of sequencing

3. sequencing bias
4. alignment
5. pairing[10] of sibling reads

The first three point are not well know, so it is assumed that all is random; the last two points depend on the programs used for alignment and subsequent pairing[11]. This usually leads to many false positives: the problem is that they are not shown, because:

- tools for structural variation detection are tested on real data, for which the structural variations are unknown; the only touchstone is the prediction of another tool, but all tools use the same idea, so both find more or less the same false positives but do not recognize them as false positives.
- sometimes simulations are run to test the tool, but
  - simulations are not very close to real life, for example because only deletions of a certain length are added or
  - simulations are run a lot of times with different parameters and only the best result is shown

  so no false positives appear.

It seems that currently used tools have no false positives, however the tests done are not suitable to allow this conclusion.

Another problem is the zygosity: this aspect of structural variation is nearly always forgotten. The problem is that for zygosity computation the exact position of breakpoints must be known and this happens only with incredible short libraries, a very rare case. So the common solution is to simply forget about zygosity.

At the moment of writing, there are five frequently used tools:

[10] in a paired-end and mate-pair library, every read is paired with another (they are 'sibling reads'); usually each read is aligned alone, then all the alignment of both sibling reads are considered and paired to form all the possible combinations

[11] sometimes a read is aligned and immediately after paired; other times, all reads are aligned, then all are paired: in both cases, pairing of a certain read happens after alignment of that read

- VariationHunter
- HYDRA
- MoDIL
- PEMer
- SVDetect

1.6.1 VariationHunter

VariationHunter[5] uses two different algorithms for predicting all the structural variations between a paired-end library and a reference genome:

- the first algorithm aims to obtain the most parsimonious mapping of paired-end reads in regions potentially having a structural variation; it is called 'VariationHunter-SC'
- the second algorithm aims to compute the probability of each structural variation; it is called 'VariationHunter-Pr'

The idea used is the same: a wrong paired-end means a structural variation, so the first thing to do is to recognize wrong paired-ends. In order to do this:

1. paired-ends are aligned against the reference
2. at the core there is the computation of the distance between sibling reads in the query genome; this distance is within a certain range and the two reads must have certain orientation
3. all alignments of a paired-end are analysed; if both reads are within the right range and orientation, they are considered correct and the alignment is called *concordant*
4. a paired-end that has no concordant alignment is called *discordant*: it can indicate a structural variation
5. a discordant paired-end can have multiple alignments, so VariationHunter chooses among these alignments the correct[12] one:

[12]where 'correct' does not mean that it is at the expected length and orientation

- according to VariationHunter-SC, the correct alignment is chosen in such a way that the minimum number of structural variation is implied
- using VariationHunter-Pr, the probability of each structural variation is computed and, according to this probability, the correct alignment is chosen

6. after correct alignments are chosen and the structural variations that they imply are found, breakpoints are considered, but only an estimation of their position is given

To test VariationHunter-SC and VariationHunter-Pr a paired-end (whole genome shotgun) library sequenced with Illumina was used. It is important to mention some features of this library:

- read length was form 36 to 41 bp
- insert size was 200 bp, so the distance between two sibling reads ranges from 164 and 159 bp
- standard deviation is about 13 bp
- sequence coverage is about $42x$
- physical coveraga is about 120

So, the library is very short and the coverage is very high: it is high costing. A lot of structural variations in this individual genome sequence were already found and validated using 40 Kbp fosmids; also with the method used by Bentley et al[6] a list of structural variations for comparison was created.

VariationHunter predicts thousands of structural variations; of all these, only some dozens overlap with structural variations found using fosminds or Bentley et al. In the discussion, only the sensibility of the tool is discussed and, although having a library with higher coverage that fosmids, VariationHunter still misses some structural variations. All the structural variations found only by VariationHunter are due to Alu elements and other trasposon, according to who creates and tested VariationHunter. Also, according to them VariationHunter is shown 'to be efficient and reliable.'

1.6.2 HYDRA

HYDRA[7] is, as always, based on the idea that a wrong arc is caused by a structural variation, so its main point is to find out all the wrong arcs. To do this:

1. low quality sequences are cut away
2. remaining sequences are aligned with BWA: it provides a good sensibility with little computational efforts
3. sequences not or wrongly aligned are realigned with NOVOALIGN, that is more sensible than BWA
4. from discording matepairs (i.e. sibling sequences with wrong distance or strand), structural variations are computed
5. matepairs forming structural variations are aligned with MEGABLAST to eliminate all false positives

When all wrong arcs are found, they are clustered together to find structural variations; it is important to mention the fact that HYDRA uses a similar clustering strategy to VariationHunter-SC. Finally, if the number of of matepairs in a cluster is enough, breakpoints are computed. The computation uses sequences from a fragment library: among the tool discussed here, HYDRA is the only one to a hybrid approach to detect strucutural variation. Using paired-end or mate-pair libraries, structural variations are found then, using fragment libraries of long reads, breakpoints of previously detected structural variation are computed. It is a good strategy, because each data type has drawbacks that are softened by the other data type, but it is a very expensive approach, because two libraries are needed, so a double amount of money to spend.

HYDRA was tested on two inbred mouse strains to see what it computed; one the results was that assemly errors are a major source of false positives in structural variation detection. Then VariationHunter was run on the same mouse data and results from the two tools are compared:

- HYDRA found 6331 structural variations
- VariationHunter found 6366 structural variations

- 6070 structural variations were in common

HYDRA and VariationHunter found roughly the same structural variations on the same data, so HYDRA works well. However, these two tools share the same clustering strategy, that is the core in their structural variation detection: using the same strategy, it's obvious that one must obtain same results.

To sum up, HYDRA uses a interesting strategy to find structural variations, but several drawbacks ruin it.

1.6.3 MoDIL

MoDIL[8] (Mixture of Distributions Indel Locator) was a tool born to detect structural variation between $10 - 50$ bp, so it focuses on the range length of structural variations that all tools (partilly also SV_finder) forget. In general, it is assumed that a small deviation of a mate-pair length is due to variance in DNA fragment size; in MoDIL the idea is that a small variance of many mate-pairs is indicative of an indel.

MoDIL uses the comparison between observed and expected distribution to detect deletions and insertions:

- expected distribution is the distribution of the library
- observed distribution is made by all the mate-pair overlapping a particular genomic location i
- the comparison between distribution is done using the kolmogorov-Smirnov test[13]
- if in i there is an homozygous indel, observed distribution shifts
- if in i there is an hetorozygous indel, about half mate-pairs shift while the other form a distribution overlapping the expected one

MoDIL was tested with simulations and real data: in simulations MoDIL was the only tool able to detect indels of range $15 - 40$ bp; real data used is very similar to that used for VariationHunter (section 1.6.1), so very high

[13]it quantifies a distance between the empirical distribution functions of two samples, in this case expected and observed distribution of mate-pair length

sequence coverage and very small standard deviation, and again very small indel was detected.

Although the conclusion of such tests is that 'MoDIL accurately recovered smaller variants than was previously possible using high clone coverage of short-read sequencing technologies', there are severe limitations:

1. only deletions and insertions can be detected; inversions are not considered
2. only insertions shorter than the library average length can be found, because longer insertions cannot produce any observed distribution
3. very small libraries are needed to find very small indels, because standard deviation must be the lowest possible, so the library must be short[14]
4. in order to have a meaningful kolmogorov-Smirnov test, a lot of mate-pairs are needed
5. consideratios 2 and 3 mean that only very small insertions could be detected
6. consideration 3 and 4 mean that only short highly sequenced libraries can be used and they cost much money

So, MoDIL is a great tool to detect indels in that length range forbidden to most tools, but is has a lot of drawbacks that heavily limit its usage.

1.6.4 PEMer

PEMer[9] (Paired-End Mapper) is a tool for the mapping of structural variation at high resolutions; it uses paired-end sequences as primary source in structural variation detection. The strategy used by PEMer is showed in figure 1.2.

To properly parameterize PEMer, simulations were used. Real chromosomes (human chromosome 2), realistic paired-end fragment size distributions, different types and length of structural variations were used to simulate data for PEMer, but:

[14] in general, the longer the library the bigger the standard deviation, and viceversa

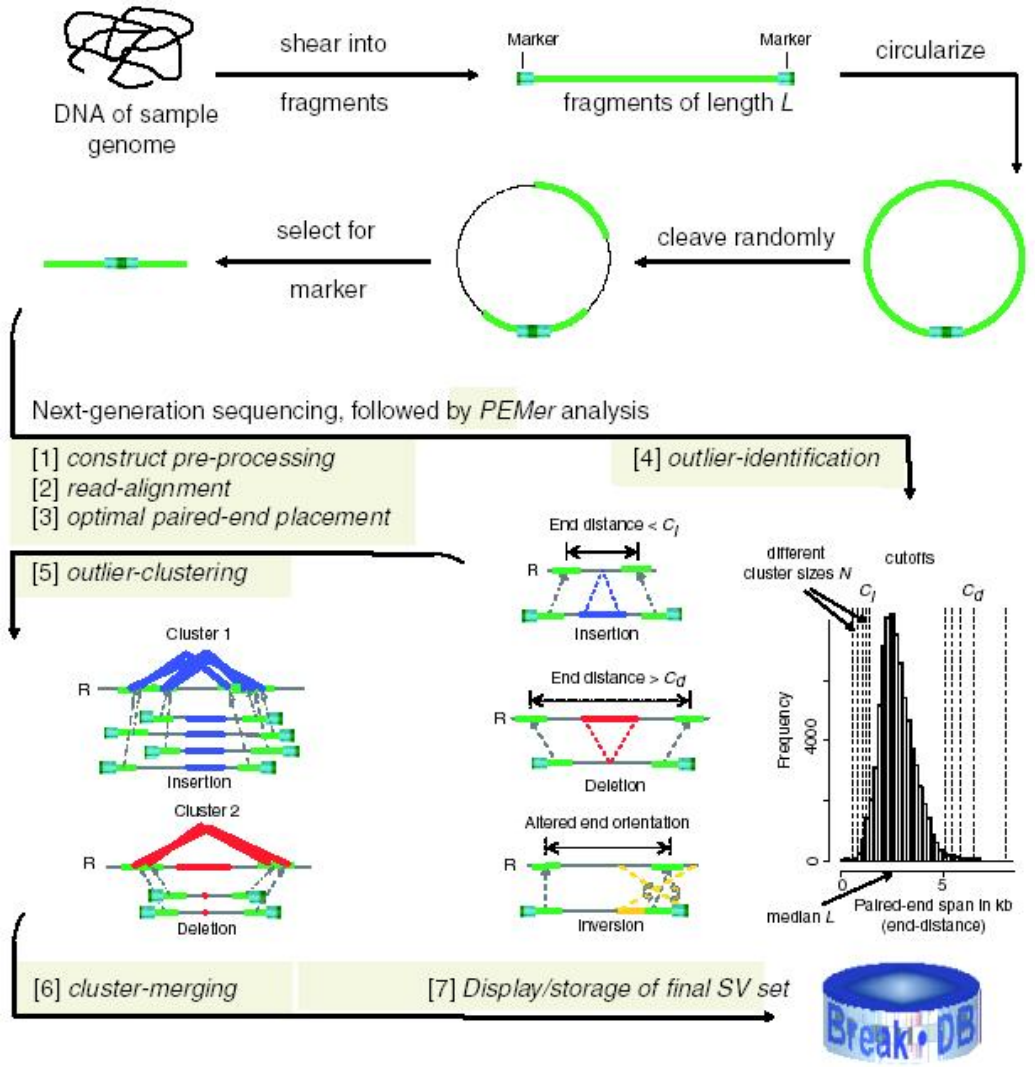

Figure 1.2: The strategy of PEMer: basically, all wrong paired-ends are considered and then they are clustered together. In case of more libraries used, in a second stage all clusters from different libraries are clustered together. Finally, all structural variations found are saved in database for further analysis.

- structural variation length was fixed, for example deletions were long

only 1, 2, 3, 4 up to 10 Kbp **in the same simulation**

- always a fixed zygosity
- the same type of structural variation in the same simulation

All simulations cover a lot of different situations, so it seems that results from simulation are very good, but in practice there is a problem, because the single simulation is a very specific situation: how many times a genome is affected only by hetorozygous deletion of 2kbp length (for example)? For SV_finder, a single simulation has all types of structural variations, with a great range of length and random zygosity. PEMer was also benchmarked against a real data alrady published; it was parameterized so new structural variations would be found, and so it was happen: a dozens of new events were found, but always these events were indel thousands of bases. Moreover only results for deletion are showed: there are no results for insertions and inversions.

A very important parameter is the span coverage, that was set to 5: it means that a $5x$ physical coverage is enough for PEMer to detect structural variations. Again, it seeme a very good results, but the problem is that only long (greater that 3 kbp) deletions are detected with such low physical coverage.

PEMer has a feature that would make it very useful: it can use data from different libraries, so in theory it is possible to have results from different libraries supporting each other. Unfortunately, this useful feature is shadowed by a nasty requirement, i.e. a high computational time: on a super computer with approximately 400 CPUs, two days of computation are required to map structural variation in a single individual. Such powerful computer is not available to all.

PEMer seems a good tool for structural variations, with interesting feature, but it requires an impossible computation power, so its use is greatly reduced.

1.6.5 SVDetect

SVDtect[10] (Structural Variation Detect) uses a sliding-window and a clustering strategy to predict structural variations (large deletions, large insertions, inversions, duplications and interchromosomal traslocations) from anomalously mapped sibling reads; SVDetect works as follow:

1. genome is partitioned into small overlapped windows of fixed size
2. anomalously mapped pairs are re-mapped to all possible window pairs, thus forming links
3. duplicated links are filtered out; coordinates, paired-end read order and orientation are defined
4. links are filtered and clusters are called based on number of pairs, orientation and order of reads, insert size threshold
5. structural variations are detected
6. a list of significant structural variations is created in proper format
7. in case of multiple data, steps from one to five are repeated for every data and results are compared to find data specific and common structural variations

As stated in its article[10], SVDetect has some 'unique' features:

- both paired-end and mate-pair library can be used
- sibling reads with a unique alignment are used to improve structural variation detection
- tandem duplication are predicted and it can distinguish between balanced and unbalanced rearrangements
- it can compare structural variations between multiple samples
- it creates copy number profiles
- it can give in output different formats to visualize structural variations found.

Only the third feature is really interesting: not many tools detect duplications, for example; the other features are used by many other tools.

SVDetect was tested with Illumina and SOLiD mate-pairs: in case of illumina, a sample of neuroblastoma cell lines was used and some cluster specific of neuroblastoma was found to show how well SVDetect works; in case of SOLiD, a wild-type and mutant strain of yeast was used and the

results were compared with another variation detection tool, GASV (no references). Other details on these comparison are not shown, so it is unclear how SVDetect really works or not. SVDetect seems a structural variation detection tool that do not shine compared to other tools.

Chapter 2

Materials and Methods

2.1 Overview of SV_finder

SV_finder is the C++ script that I develop during my PhD in order to find out structural variations affecting a sequenced genome. C++ was the language used because the first problem to solve was the development of a **fast** program able to handle a lot of data in reasonable time. NGS output consist of millions and millions of sequences, and each of them needs to be aligned against a reference genome and classified by its length, orientation and number of hits. After these two steps, SV_finder takes certain classes of reads and 'translate' them into a list of structural variations. Moreover, a library can be sequenced more than one time, so its possible to have more than one run as input; SV_finder can handle input from more sequenced libraries, even from different libraries if their average length and standard deviation are not too much different, i.e. their peaks are almost overlapped. Relative fastness and much highly memory required was the main reasons to make me chose C++ language.

SV_finder can work with base space and color space, i.e. it can use data from SOLiD and from Solexa; obviously, also 454 output is a possible input data, but its low output with paired-end and mate-pair library is not suitable in order to find efficiently structural variations. SV_finder can use more than one run; actually, it is not designed to handle runs of very different libraries in order to improve results by analyzing arcs of very different length, but this limitation is overcome by some improvements so it can achieve good results even with one single library. SV_finder can use every known genome in its

algorithm: all the required files and variables are parametric.

The output consist of three files, each for every type of structural variations; these files consist of a list of the corresponding structural variations found, mainly defined by a position on the reference genome. According to some results of real data (see section 3.3) a human genome suffers from thousands of structural variations; these can be seen as too much data and as too little data. A list of thousands of entries has a lot of 'background noise' if one is interested only to a particular gene, region or position into the genome, so for him the output is too much and he needs something to help him focusing on right target. On the opposite case, one can be interested into finding all structural variations affecting a genome, but so far it is unknown how many of them are present in a genome, so a thousand of structural variations ca be a poor number. It is clear that the final list of structural variations is **not** the final step in serious analysis, but it's the first step: after the prediction of structural variations, one have to study in details each on them. It's the same concept for a sequence genome: it's very important to sequence an unknown genome (as human genome before 2003) in order to get better insight of it and solve problems and diseases affecting it; but it's not enough to have the raw sequence, it's necessary to use the raw sequence as base for further studies.

2.2 Input Data

SV_finder requires some files and parameters to work. Required files include reference in fasta format, aligned and classified arcs in gff or sam format, sequenced reads in fastq or csfasta format and sequenced reads that not align in fastq format. Required parameters include tags for identifying forward and reverse read in a pair, space of sequence (base or color), output directory, consent to store all information in RAM memory instead of analysing chromosomes one by one and consent to store as files all the spliced alignments done by SV_finder. All of these are the parameters required at least to make the program work; after them, the program may need other parameter in order to adjust results or speed of the structural variation prediction. These parameters are mainly numbers defining how to create structural variations in the first step, numbers defining how to make spliced-alignment and numbers defining how to estimate zygosity of structural variations in the second step. The user may define all these numbers, otherwise there are default

values that can be used; default values are optimized parameters for the majority of cases.

2.2.1 Formats

SV_finder can work with different formats. Fasta format are always required for the reference, and it is very simple: first line has name of the reference preceded by the '>' symbol, while other lines has the reference sequence, in general with 60 characters per line. In case of more chromosomes present, each chromosome reference is described as above and they are simply queued.

Alignments are in sam or gff format; both are tabular file composed of a certain number of fields. Meaning of fields includes name of aligned sequence, chromosome where the sequence align, alignment position (start and end), alignment strand, potential mismatches or gaps, plus other miscellaneous informations.

Sequenced reads are in fastq or csfasta format. A fastq file uses 4 lines for every read, each one containing respectively: name of the sequence preceded by the '>' symbol, sequence itself, separator character, quality of the sequence. A csfasta file is used for sequence in color space and uses 2 lines for every read, containing respectively name of the sequence (preceded by the '>' symbol) and sequence itself.

2.2.2 Alignments Organization

The most important information needed is obviously the alignments of the sequenced library. They have to be organized in a certain way to make SV_finder work. All the alignments have to be classified according to distance, orientation and hit number of every couple of reads. SV_finder focuses only to read uniquely aligned; only for them one can be sure about their position on the reference, and position is the most important source of evidence for structural variation, so if a read is aligned many times (i.e. in many different positions), one cannot know for sure which position is the right one, so it's impossible to detect which alignment is 'right' (i.e. not interesting to detect structural variations) and which is 'wrong' (i.e. very interesting to detect structural variations).

SV_finder uses four different files as source of alignments to use; these files must be named and contain certain reads as follow:

UNIQUE_PAIR : reads forming an arc that has right length (L_r) and orientation; 'right length' is defined as:

$$AvgLen - 3 * StdDev \leq L_r \leq AvgLen + 3 * StdDev$$

AvgLen e *StdDev* are respectively average length and standard deviation of distance distribution (see section 1.5).

Right orientation depends from the library, but in general paired-end library has sibling reads that must align with different orientation, while mate-pair library has sibling read that must align with the same orientation; other combination of orientation are wrong; orientation is usually pictured with a '+' or '-' symbol so, for example for paired-end library, the two reads that form an arcs must align one on '+' strand and one on '-' strand.

UNIQUE_WRONG_D : reads forming an arc that has wrong length (L_w) but right orientation; 'wrong length' is defined as:

$$L_w \leq AvgLen - 3 * StdDev \vee AvgLen + 3 * StdDev \leq L_w$$

AvgLen, *StdDev* and right orientation are the same as above.

UNIQUE_WRONG_S : reads forming an arc that has wrong orientation; length doesn't matter (see section 2.4.3).

UNIQUE_SINGLE : reads which sibling is sequenced but is not aligned; because in the pair only one read is aligned, arc doesn't exist, so length and orientation don't have any meaning.

An arc is defined by two reads, the siblings of a pair; so, for all files except UNIQUE_SINGLE, odd lines the read that define the first part of an arc, and the following line has the sibling read that define the second part of the arc. For this reason, UNIQUE_PAIR, UNIQUE_WRONG_D and UNIQUE_WRONG_S files are read two lines at once.

UNIQUE_SINGLE file has no arc, so every line has simply one read aligned.

2.2.3 Color Space

SOLiD NGS introduced the special feature of color space: instead of producing sequences in base space, i.e. made up of A, C, G and T nucleotides, it produces sequences in color space, i.e. made up of 0, 1, 2 and 3 numbers. These numbers arise from two consecutive nucleotides; in order to translate a color space sequence into a base space one, the first nucleotide must be known so, according to it, the remaining colors has a unique meaning (see section 2.5.4 for a better explanation).

2.2.4 Reads not Aligned

Insertions are a bit different from deletions and inversions, mainly because there is a novel sequence not present in the reference; arcs that have at least one read inside the insertion behave in a strange way:

- arcs with one read inside the insertion and its sibling outside drive to signal for long insertion (see section 2.4.4)
- arcs with both reads inside the insertion don't align at all

There is another reason to make two sibling reads not align: one read is inside the insertion, its sibling cover one of the two breakpoint of insertion itself. In this case, the read covering the breakpoint can be splice-aligned onto the reference, thus identifying the breakpoint (see section 2.5.3 for a better explanation).

2.2.5 Read Tags

In a paired-end or mate-pair library, every read has a sibling and the two are matched; they arise from the same DNA fragment and are its sequence ends. But it's important to distinguish between the two ends, two matched reads share the same name but a different tag. These 'tags' often are '1' and '2', or 'forward' and 'reverse'. SV_finder needs the tag to know which reads can be splice-aligned in order to find the breakpoints (see section 2.5).

2.2.6 Alignment Word

These is a very special parameter needed for the splicing alignment that is not estimated by default. Basically, when a spliced alignment is computed,

the program decides to keep or throw away it by some parameters, as number of allowed mismatches at end of a read, minimum percentage identity and minimum number of matches. These parameters are a double edged sword: if they are too relaxed, a lot of spliced-alignments are found, so the number of false positives dramatically increases and the the breakpoint are recognized with a high error; if they are too strictly, a lot of spliced-alignment are missed, so sensibility suffers a lot. In order to avoid these two situations, the three previous parameters are kept relaxed and an **alignment word** is introduced to avoid non-sense spliced-alignments. This word tells the program how many mismatches are allowed for alignments of a certain length.

2.3 Two Steps

SV_finder is divided into two main steps. The first one takes all the 'wrong' arcs in order to detect the regions in the reference genome that can be affected by a structural variation, while the second one takes such regions and try to spliced-align in them certain reads in order to both back up the region and find the breakpoints. Finally, the zygosity of all structural variations found are estimated.

The reasons for such a division arise from how first and second step work in order to improve performances of the program. Basically, first step is similar to what other tool do (see section 1.6), so 'wrong' arcs are taken and are clustered together to identify a structural variation. The differences between other method and SV_finder mainly concern three thing:

1. how do I consider an arc as 'wrong'?
2. how do I cluster together 'wrong' arcs?
3. how do I translate clusters of 'wrong' arcs into structural variations?

Other methods choose different ways to answer to the above questions, sometimes in a very advanced way, while SV_finder uses a very simple way to detect structural variations in the first step. The main problem of this approach (i.e., taking all the 'wrong' arcs in order to detect structural variations) is that a structural variation gives birth to 'wrong' arcs, but a 'wrong' arc doesn't always arise from a structural variation: in some cases, 'wrong' arcs are caused from sequencing errors, assembly errors of reference, errors or bias in library preparations, alignment tool used, parameters used. This leads to

a very high number of false positives. Unfortunately, all the aspects are still almost unknown, so it is not known how they contributed to 'wrong' arc. Moreover, if the distance distribution of sequenced library has a great standard deviation, also the 'error' of each arc is great, so the breakpoints are not well detected. The consequences are serious: in order to detect breakpoints with a good precision, the sequence library must have low standard deviation; because standard deviations is linked[1] to average length, the library must be very short in order to keep error on arcs low. But a short library has short arcs, so their number must be high in order to have a reasonable number of arcs covering each positions, and this means a lot of money to spend in order to sequence a lot: a short library translate into more money needed. Furthermore, the number of false positives remains high: no matter how a program works, how well the library is made, how short the library is, there will always be wrong arcs not dependent form structural variations, and these arcs will lead to false positives.

SV_finder introduces a second step in order to solve the two main problems: false positives and poor breakpoints resolution. In the second step, certain reads are considered and splice-aligned on the structural variations found in the first step. This allow to confirm the structural variation itself by providing a second evidence beyond the wrong arcs present; moreover, splice-alignments happen at base-level, so they detect breakpoint with a base-precision, regardless of library standard deviation.

Finally, the availability of precise breakpoints allows an estimation of zygosity of each structural variations found, something that most of the already existing methods seem to forget. To estimate zygosity, arcs covering only one breakpoint and arcs covering the entire structural variation are counted; zygosity depends from which type is the most numerous. This estimation is very good in case of structural variations longer than the library[2]; when the structural variation is roughly equal or shorter than the library, the different types of requested arcs are not so different, so their counting can be dubious, leading to errors in the final estimation.

[1] standard deviation in linked to average length mainly for library preparation: when the DNA is fragmented, the fragments are separated by a electrophoresis gel and the fragments of wanted length are selected; however, the separation is not very perfect, so it is possible to obtain only fragments of a certain range of length rather of precise length; for this reason, in general standard deviation is about one tenth of average length, so the bigger the library, the greater the standard deviation

[2] the size of a library is its average length

2.4 First Steps

As stated in previous section, first step takes all wrong arcs and, based on them, detect the structural variations. Different types of arcs are connected to different types of structural variations:

deletions : a deletion gave birth to arcs that align at longer distance than library average length; basically, all arcs that cover the deletion point in the query genome align onto reference with right orientation and great distance

insertion : an insertion can gave birth to arcs that align at short distance than library average length; if the insertion has a length that allow arcs to cover the entire insertion in the query genome, these arcs will align onto reference with right orientation and small distance. If the insertion is to long, such arcs don't exist. Also, some arcs has a read outside the insertion and a read inside: the longer the insertion, the bigger is this number of arcs. For these arcs, only the read outside the insertion align onto the reference.

inversion : an inversion give birth to arcs that align with wrong orientation[3]; all arcs that have a read inside the inversion and a read outside, align with wrong orientation

In section 2.2.2 the files with the source of alignments are explained; those 4 files contain the arcs cited above, in particular:

UNIQUE_PAIR : has arcs useful to detect short deletions and short insertions

UNIQUE_WRONG_D : has arcs useful to detect long deletions

UNIQUE_WRONG_S : has arcs useful to detect inversions

UNIQUE_SINGLE : has arcs useful to detect long insertions

'Short' and 'long' refer to the library average value: short indels have a length smaller (or comparable) than the library average value, while long indels have a length greater than library average value.

[3]wrongness and rightness for orientation depend from how the library was made

Structural variations are divided into 5 classes: short deletion, long deletion, short insertion, long insertion, inversion. Each class are found by the analysis of one of the previous 4 files. Such analysis give birth to 5 array of values, with a value for every position, so the length of each array is equal to the length of reference genome. Using a threshold value (specific for each array), these arrays are translated into structural variations by using contiguous values equal or above threshold value. Structural variation found depends from type of array.

2.4.1 Observed Distribution vs Expected One

Short indels can be found by the analysis of those arcs that seem to have wrong length. A wrong length occurs when the observed length is different from the expected one. Expected length al_e is the average length of all the arcs: it is computed at the beginning, once for every position, by taking all the arcs from the UNIQUE_PAIR file, making them form the expected distance distribution and calculating its average value. Observed length is more tricky: for every position, all the arcs covering it form an observed distance distribution; its average length is the observed one. For a position i, an arc A connecting position j to position k covers i if:

$$j \leq i \leq k$$

All arcs A form the observed distribution for position i. The average length al_o of observed distribution is:

$$al_o = \frac{\sum_{k=1}^{N} A_k}{N}$$

where:

A_k is the length of k-th arc

N is the number of arcs

The value given to position i is saved in position $i-1$ on the right array and is:

$$al_o - al_e$$

This value (figure 2.1) can be positive, negative or zero:

positive : in case of deletion, all positions covered by one or more wrong arcs have a positive value because their observed distribution are shifted to the right compared to expected distribution; so, positions flanking and inside the deletion region share positive values; positive values are saved in array 1a, while array 1b takes a zero value in the same indexed position

negative : for an insertion, all positions covered by one or more wrong arcs have a negative value because their observed distribution are shifted to the left compared to expected distribution; so, positions flanking point of insertion share negative values; negative values are saved in array 1b, while array 1a takes a zero value in the same indexed position

zero : if there is no indel, observed distribution and expected one are equal, so there is no shifting and their average values have no difference, in theory; in practice, arcs share a certain degree of randomness, so almost never the value is exactly zero, but often very close to zero; a zero value is saved in both arrays 1a e 1b

The computation is done for every position of the reference genome by taking all the arcs from file UNIQUE_PAIR and UNIQUE_WRONG_D of length L as

$$0 \leq L \leq 2 * AvgLen^4$$

After this analysis, array 1a and 1b are filled.

Weight Average Length

The average length of expected distance distribution is computed in two different ways: as an average value of a distribution (like al_o) and as a weighted average value of a distribution. The first way lead to computing what I called $AvgLen$, and this value is the value always used except for calculating values in arrays 1a and 1b. The second way leads to computing the al_e used in 2.4.1:

$$al_e = \frac{\sum_{k=1}^{N} L_k^2}{\sum_{k=1}^{N} L_k}$$

[4] $AvgLen$ is the average length of the expected distance distribution and is a bit different from al_e (see section 2.4.1)

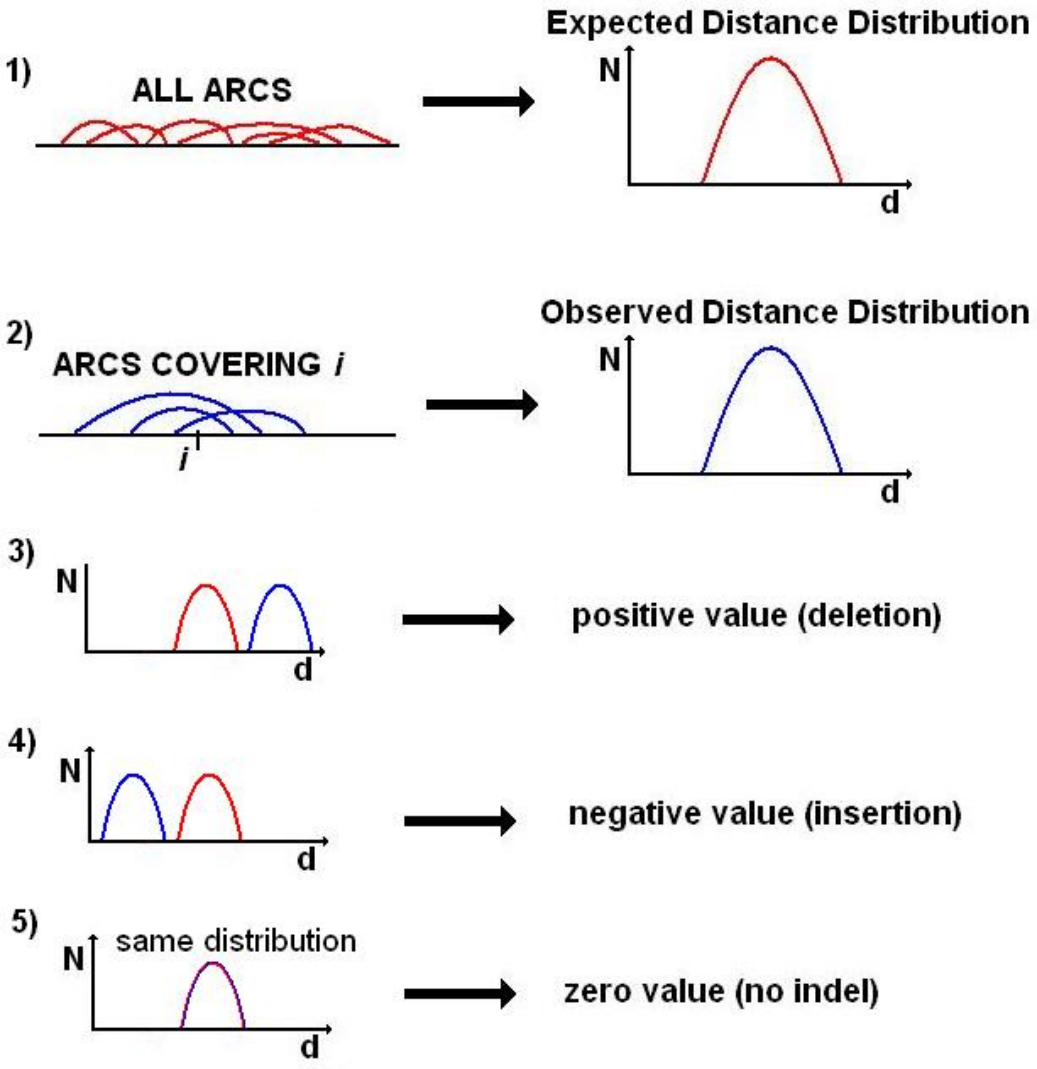

Figure 2.1: The computation of arrays 1a and 1b; in 1 all the arcs are taken to compute the expected distribution (red line); in 2 a position i is considered, all arcs covering i form the observed distribution (blue line); in 3, 4 and 5, expected and observed distributions are compared each others to compute array values.

where:

L_k is the length of k-th arc

N is the total number of arcs

Basically, the weighted average value of the length of arcs is computed using the length itself as weight; the difference between al_e e $AvgLen$ is very small, about 5% in favor of al_e. When observed distribution are considered position for position, it happens that an arc appear in as many positions as its length: the longer the arc is, the more times it is used in computing a al_o. This lead to having long arcs considered more than short arcs. When $AvgLen$ is computed, every arc is taken only one times, regardless of its length, so this effect of overweighting long arcs do not appear: using $AvgLen$ to compute the difference between observed and expected distance distribution lead to overestimating the shifting (see figure 2.2), thus predicting deletions where nothing are present and skipping insertions with a small shifting.

Using al_e, all the arcs are weighted by their length, so long arcs count more than short arcs: the computing of observed and expected average length has no difference in concept. In this way, the shifting is not overestimated and deletions ad insertions are rightly predicted (see figure 2.3).

Limitations

The comparison between observed distribution and expected one has a major limitation: only short indels can be found, i.e. indels long as or shorter than library average value. For insertions, the explanation is simple: an insertion longer than the longest arc cannot have arcs entirely covering it. If it is too much long, during the library preparation it is impossible to obtain a fragment longer than the inserted DNA, so it is impossible to have an arc with a read flanking the insertion on the left and its sibling flanking the insertion on the right. From a mathematical point of view, in case of insertion the observed distribution shifts on the left respect to the expected distribution; this shifting has an end, represented by the 0: the observed distribution can shift until it reaches the 0 point, which means that it is made up of arcs with no length[5]. This situation corresponds to insertions long as the possible longest arc. Having arcs even longer means having observed distribution with negative values, but this is impossible, because a DNA fragment cannot have a negative length. For deletions there is no such constraint: a deletion

[5]an arc with no length happens when its two reads align adjacent

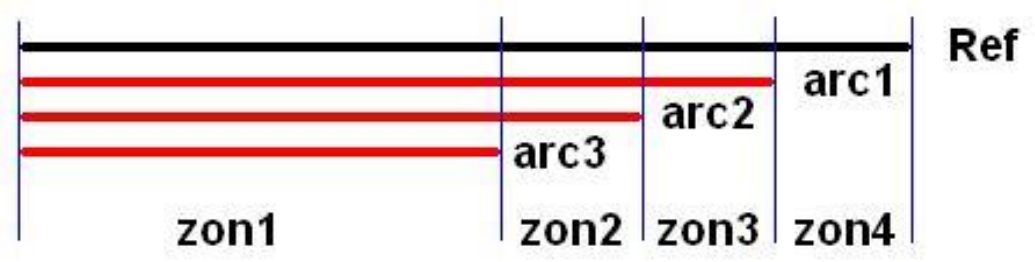

Figure 2.2: The effect of using $AvgLen$ instead of al_e. In the example there are 3 arcs (red lines) aligned against the reference genome (black line marked with 'Ref'); no structural variations is present and $AvgLen$ is given by all 3 arcs present. Based on how arcs align, 4 different regions appear: zon1, zon2, zon3 and zon4: 1) zon1 has all 3 arcs, so their average value is equal to $AvgLen$ and the difference between the two is zero (no indel predicted); 2) zon2 has the two longest arcs, so their average value is a bit greater than $AvgLen$ and the difference between the two is a positive value (a deletion predicted with low signal); 3) zon3 has only the longest arc, so its average value is far greater than $AvgLen$ and the difference between the two is a positive value (a deletion predicted with great signal); 4) in zon4 no arcs are present, so saved value is zero (no indel predicted). Although the complete lacking of structural variations, a deletion is found in zon2 and zon3 using $AvgLen$ as average value.

make the observed distribution shifts to the right of the expected distribution, and there is no limit here. Ideally, a deletion of length L, has a shifting of L between its distribution (observed and expected). However, I choose to set a limit, and this $2 * AvgLen$. The reason is that, because the analysis is made position for position, including very long arcs means that they affect a lot of position; long arcs come from deletions but also can be simply errors, so including very long arcs can mean including error in analysis for a lot of positions. Also, I want to balance someway between insertions and deletions so, because insertions could have an observed distribution between 0 and $AvgLen$, i choose to limit the comparison of observed and expected distributions between $AvgLen$ and $2 * AvgLen$.

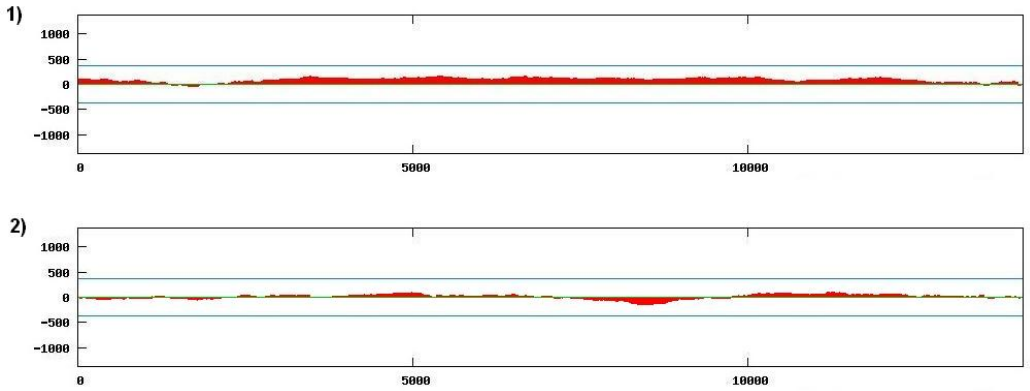

Figure 2.3: Red areas are the computed value saved in array 1 for each positions. No structural variation is present in the region showed. Saved values are the difference between average values of observed and expected distributions: 1) $AvgLen$ is used as average value for expected distribution and, as result, nearly each position has a positive value, which suggests a deletion; 2) al_e is used and now positions have a very small value, positive or negative, as expected when no structural variation is present.

2.4.2 Long Arcs

Arcs longer are analysed to find out long deletion, but no comparison of distributions is made. Instead, SV_finder takes all long arcs and compute their arc coverage. An arc A is considered 'long' when:

$$A > 2 * AvgLen$$

Arc A has a read aligning at positions i and j and its sibling at positions k and y; obviously, i and j are, respectively, start and end of the first read, while k and y are start and end of the second read. Also, it's true

$$i < j < k < y$$

In such a situation, the coverage computed using positions i and y as start and end is a physical coverage; the coverage computed using positions j and k is the arc coverage. Because I am interested about arc coverage, arc A cover a position p if

$$j + 1 \leq p \leq k - 1$$

Arc coverage is computed by taking into consideration all long arcs; for position p, its arc coverage AC is

$$AC = N$$

where N is the number of long arcs covering position p. Arc coverage for every position is saved in array 2. As stated in section 2.4.1, long arc can arise from errors rather from long deletions; by using arc coverage, a long arc influence a lot of position, as stated. But the influence is not as big as comparison between distributions: a long arc can make the average length of observed distribution become great, even if there are 1 long arc and 50 right arcs; in arc coverage, long arcs coming from errors are spread among the entire genome and it is very unlikely that they are clustered in the same regions. They form a background noise and, if threshold values are correctly set and library is enough sequenced, it's not a problem. Source of long arcs is represented by the file UNIQUE_WRONG_D.

2.4.3 Wrong Strand Arcs

When an inversion occur, arcs with a read outside the inversion and a read inside align with a wrong strand. They are saved in file UNIQUE_WRONG_S, which is used as source in detecting inversions. The method used is, basically, the same for long deletion (section 2.4.2), so SV_finder takes all the arcs present in file UNIQUE_WRONG_S and compute their arc coverage. Values are saved in array 3. In this case, there is no limitation on length of inversions: any inversion makes arcs align with wrong strand, so any inversion can be detected in this way. A problem can occur when there is a very short inversion; in fact, given

- a sequenced library that, after alignments, cover the reference genome with a sequence coverage of C
- a sequenced library of reads that are R bp long on average
- an inversion I of length L on the query genome

the average number N of arcs that we can count for every position affected or near the inversion I is

$$N = \frac{C * L}{R}$$

The consequence is clear: the signal that can be get by an inversion is directly proportional to its length; long inversions give a great signal, while short inversions give low signal. If the inversion is too short, it can happen that its signal is lower than background noise. But it can also happen that the signal is lower that the threshold value used. Lowering this value could be the solution to the problem, but lowering threshold values can lead to much more false positives. So, in case of long libraries, it can happen that inversions shorter that library average value are miss, due to the threshold value used (see section 3.1.4). For arcs with wrong strand, the length is not important: in fact, when an inversion occurs, the read inside it changes completely its position, thus totally changing arc length (figure 2.4). It's possible to compute the new length NL. If:

- there is an inversion I of length L
- there is an arc A, with a read R_o outside the inversion and its sibling R_i inside; R_o and R_i are on left side of inversion in the query genome
- R_i is at d_i bp from the left breakpoint in the query genome
- R_o is at d_o bp from the left breakpoint in the query genome

the original[6] length of arc A is $d_o + d_i$. In the reference genome, all that is inside the inversion is, obviously, inverted, so the new length of A will be $d_o + L - d_i$. One needs to know L, d_o and d_i, which means knowing length of inversion and its breakpoints. But these two informations are unknown before the analysis takes place; when computing values for array 3, I don't have any informations to know the real length of the arc, so I just forget about it.

2.4.4 Reads without Aligned Sibling

SV_finder makes goods use of a very special type of reads; an arc is made up of two sibling reads aligned that, thank to the alignment itself, connect

[6] 'original' means 'in the query genome'

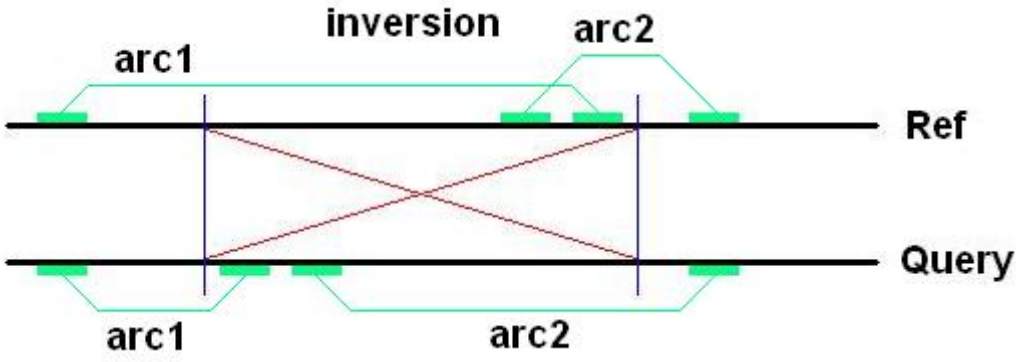

Figure 2.4: Arcs with a read outside and its sibling inside an inversion change their length: arc1 becomes longer and arc2 becomes shorter than their original length.

two distant point int the genome, creating, precisely, an arc. But what does it happen when one of the two sibling read doesn't align? No arc is created. Most of other tools for structural variation detection simply skip this; they claim that, if an insertion is too long, there is no arc that can jump over it, so the insertion cannot be detected. SV_finder uses aligned reads which sibling is sequenced but not aligned in order to detect such too long insertion. When an insertion happens, certain arcs has one read outside the insertion and its sibling inside. Inside reads cannot align into reference genome, but outside reads align and form a scattered region of reads around the insertion point (figure 2.5). The length of this region is directly proportional to average value and standard deviation of distance distribution. Single reads are contained in UNIQUE_SINGLE file.

The scattered region can be detected by running a window over the reference genome and counting the number of single reads aligned inside the window. The length L_w of this window depends from the distance distribution and is

$$w = AvgLen + 3 * StdDev^{7}$$

where

[7] I used $3 * StdDev$ because at 3 standard deviation from the average value fall 99.6% of all values in a gaussia distribution; see section 1.5

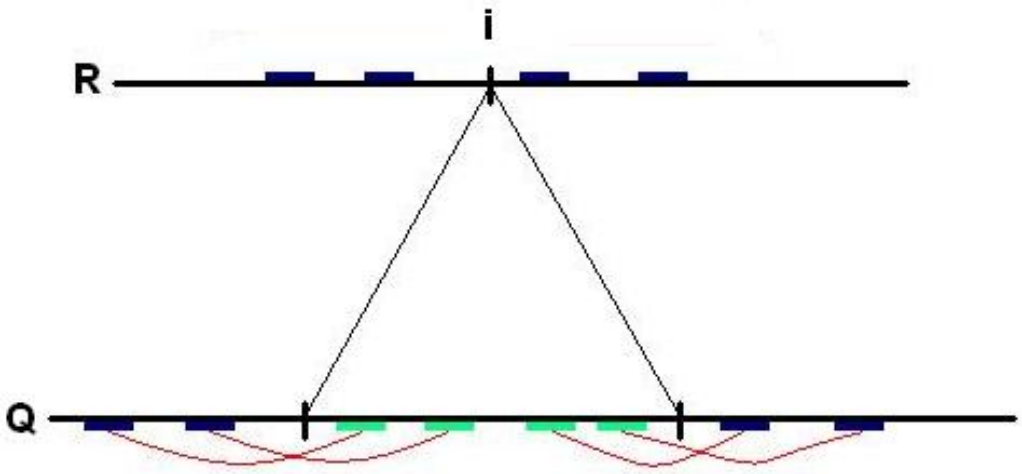

Figure 2.5: In case of insertion in the query genome ('Q'), blue reads are outside the insertion and green reads are inside; only blue reads align against the reference genome ('R'), thus forming a scattered region of reads around the insertion point i.

- $AvgLen$ is the average length of distance distribution
- $StdDev$ is the standard deviation of distance distribution

For a position i, the window w cover a position p if

$$i - w \leq p \leq i + w$$

Every single read with right direction that align within w is counted. Total number of single reads N_i in w is given as value to position i (figure 2.6). Values for every position of the reference genome are stored in array 4.

The problem with this signal is that scattered region is very large: in total, it is $2 * (AvgLen + 3 * StdDev)$ bp long, because it is possible for a read to be at $AvgLen + 3 * StdDev$ bp from the insertion point, both on the right and on the left. Some examples better explain the problem:

- library with 500 bp of average length and 50 bp of standard deviation: scattered region is $2 * (500 + 3 * 50) = 1300$ bp
- library with 1600 bp of average length and 300 bp of standard deviation: scattered region is $2 * (1600 + 3 * 300) = 5000$ bp

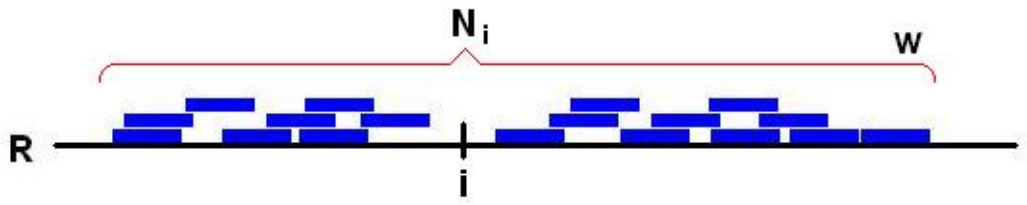

Figure 2.6: All single reads within window w are counted and their number is the value assigned to i in array 4.

- library with 8000 bp of average length and 1000 bp of standard deviation: scattered region is $2 * (8000 + 3 * 1000) = 22000$ bp

Even with a small library ($AvgLen = 500$ bp and $StdDev = 50$ bp) the scattered region is large: identifying an insertion with a thousand base of error is very bad (figure 2.7).

All insertions can be found by counting single reads, even short insertions found by comparison between observed and expected distributions; the problem for short insertions is similar to that for short inversions: the shorter the insertion is, the lower the number of single reads is, so it is possible to skip the insertion just because a too great threshold value. Unlike for inversions, this doesn't lead to severe consequences: there is a way to detect short insertions (see section 2.4.1), so SV_finder doesn't risk to skip them.

Parameters

Unlike the previous signals, this analysis can be affected by two parameters, '-alone' and '-direction'. A certain number of reads doesn't have a sibling sequenced; these reads do **not** suggest insertion; they have the same property of a fragment libraries, so are not very useful in detecting structural variations. Because of this, single read without a sibling sequenced form an annoying background noise and thus it is better to skip them. Parameter '-alone' take all reads in UNIQUE_SINGLE file and compare theme to all sequenced reads; a read without its sibling aligned is skipped.

A single read can have its sibling on the left or on the right of its position. This depends from the library type. As default, SV_finder doesn't know

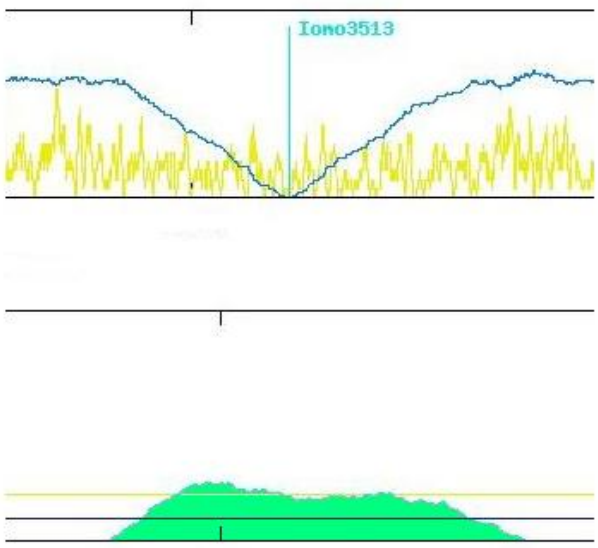

Figure 2.7: An insertion of 3513 bp produce a scattered region of about 4000 bp! Between these 4000 nucleotides there is the right breakpoint. The library used has an average length of 1600bp and a standard deviation of 300 bp: with longer libraries, the scattered region becomes larger.

any position for the sibling reads, so it does the analysis assuming that the sibling reads can potentially lies both on the left and on the right of single reads. This makes the scattered region around an insertion point be larger than expected, thus worsening the detection of insertions. With parameter '-direction' it is possible to tell SV_finder where to expect position for sibling reads (figure 2.8).

2.4.5 Potential Structural Variations

After the previous calculations, SV_finder has stored in memory 5 arrays containing values for each position of the reference genome. Each array is linked to a certain type of structural variation, as follow:

- **array 1a** has values useful to detect short deletions
- **array 1b** has values useful to detect short insertions

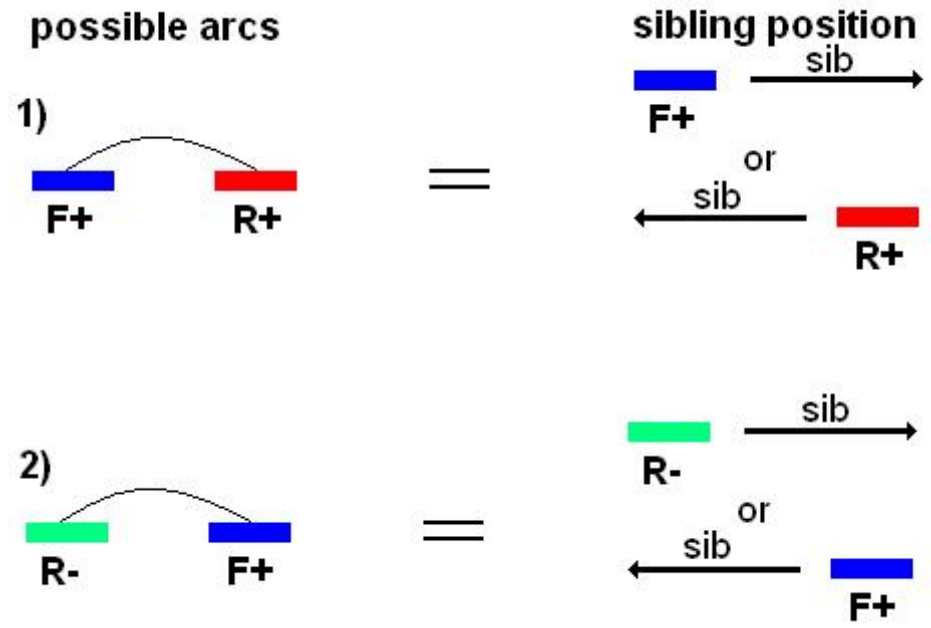

Figure 2.8: Two examples on the position of sibling read: 1) the only possible arc is made up of the forward read on the left and the reverse on the right, both on '+' strand, so when the single read aligned is the forward on '+' strand (F+), its sibling is on its right; for the R+ the idea is the same 2)the only possible arc is made up of the reverse read on the left and on '-' strand, while the forward is on the right and on '+' strand, so when the single read aligned is the reverse on '-' strand (R-), its sibling is on its right; for the F+ the idea is the same.

- **array 2** has values useful to detect long deletions
- **array 3** has values useful to detect inversions
- **array 4** has values useful to detect long insertions

Each array is translated into a list of **potential** structural variations by using a specific threshold value: contiguous positions with a value equal or higher than given threshold are saved as a structural variation, according to the array used (see figure 2.9). Only for potential structural variations of array 4, the middle point m is computed and the potential structural variation is

re-calculated as between positions $m - \frac{AvgLen}{2}$ and $m + \frac{AvgLen}{2}$. This is based upon the fact that the insertion point must be in the middle on the scattered region, so the middle point is taken as insertion point and a error equal to the average length of an arc is added.

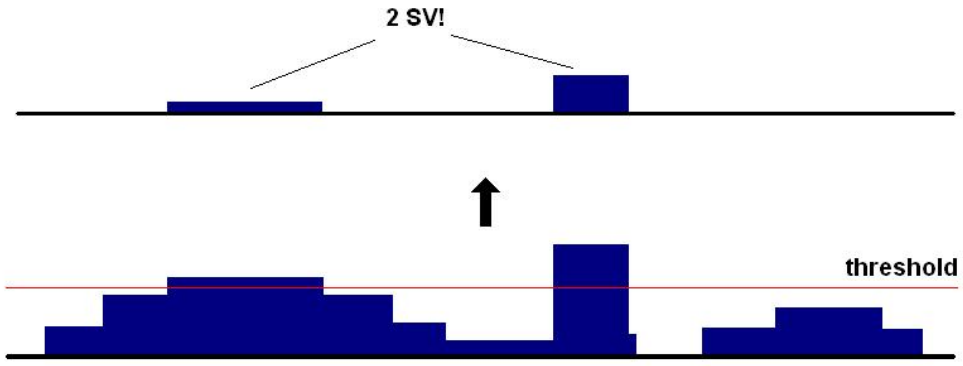

Figure 2.9: Blue area is made up of values for each positions in an array; using a threshold value (red line), all contiguous positions with a value equal or higher that threshold are translated into potential structural variations.

Threshold values are specific for each array and depend from how array values are computed.

It can happen that some potential structural variations overlap, mostly for deletions and insertions of length around $2 * AvgLen$. For these lengths, deletions give signal both for arrays 1a and 2 and insertions for array 1b and 4. In the second step, SV_finder assumes that every potential structural variation has at least one confirmed structural variation of certain type, so overlapping structural variations of the same type must be solved:

deletion : potential structural variations coming from array 1a and 2 are joined together

insertions : potential structural variations coming from array 1b and 4 are joined together

The join happens only for potential structural variations overlapping or adjacent; if potential SV $pSV1$ between positions i and j, and potential SV $pSV2$ between positions k and y overlap or are adjacent, a new potential SV replace $pSV1$ and $pSV2$ and is between positions i and y.

Why 'Potential'?

After the first step, SV_finder generates a list of what I called potential structural variations. These list could be similar to the output of other tools that detect structural variations, maybe it is worse due to the more simple method that SV_finder use, but this list from to major problem, as discussed previously:

- a high number of false positives
- poor breakpoint resolution

A structural variation found soon after the first step is often larger that the real SV, and sometimes doesn't exist at all; for these reasons, SV_finder calls all the SV found in the first step as 'potential structural variations'.

Threshold Values

The 5 arrays containing values for each position are translated into potential structural variations by using a threshold value specific for each array; because arrays 1a and 1b share the same value, only 4 threshold are needed. Arrays 1a and 1b has difference between average value of observed and expected distributions; the standard deviations of expected distribution is linked to the shifting that an observed distribution can show without an indel: about 99.6% of rigth arcs fall within 3 standard deviation from average value so, in first versions of SV_finder, the program used one standard deviation as threshold value[8]. However, it is known that standard deviation depends from the numerosity of sample taking into consideration:

- in a population of P samples, by taking one sample one by one, a certain standard deviation sd is computed
- in the same population as above, if N samples are taken, their average values is computed and then used as sample for a new population, the standard deviation of the new population will be $\frac{sd}{\sqrt[2]{N}}$

When the difference between distributions is computed, SV_finder takes all arcs covering a certain position to form the observed distribution; this is the

[8] one *StdDev* was used in replacement of 3 *StdDev* in order to do not miss too short indels

same as taking N sample at once: the standard deviation that they refer is $\frac{StdDev}{\sqrt[2]{N}}$. So, for array 1a and 1b, the threshold value depends from the arc coverage. When there are few arcs (which means N is small), the threshold is a value close to $StdDev$, because there is a great uncertainty about those arcs; on the contrary, when there are a lot of arcs (which means N is big), the threshold is a value very small respect to $StdDev$, because there is more certainty about those arcs. By default, $\frac{StdDev}{\sqrt[2]{N}}$ is taken as threshold values.

Arrays 2 and 3 have values based on arc coverage; if the average arc coverage is C, one can expect to have around C arcs covering an inversion or a long deletion; in case of heterozygosity, around half arcs will come from right chromosome and the other half from chromosome affected by the structural variation, so about $\frac{C}{2}$ arcs will be present. By default, SV_finder takes $\frac{C}{4}$ as threshold value for arrays 2 and 3: by doing this, inversions or long deletions in heterozygosity are not missed and very few false positive are introduced.

Array 4 has values coming from the counting of single reads within a certain window: count a single read is the same as count an arc, so threshold value depends from average arc coverage C, as for arrays 2 and 3. But in case of an arc, one can be sure about the positions of its sibling reads; on the contrary, in case of a single read, if not specified, one cannot know where its sibling could lie. SV_finder knowns where the sibling lie only when '-direction' parameter is set, so:

- if '-direction' parameter is set, threshold value is $\frac{C}{4}$
- if '-direction' parameter is unset, threshold value is $\frac{C}{2}$

When '-direction' parameter is set, having a single read is the same as having an arc, so for the same reasons of arrays 2 and 3 one can use $\frac{C}{4}$ as threshold value; but when SV_finder doesn't know where to place the sibling reads, each single read give a double signal, so values for insertions, errors and background noise are as well as doubled, so the threshold value too must be doubled: $\frac{C}{\mathbf{2}}$. Also, potential structural variation indicating a long insertion will be larger with '-direction' parameter unset (see section 2.4.5).

Parameters

Computing potential structural variations can be influenced by the user in many ways, mostly by using threshold values different from default ones;

also there are two more parameters to help avoid false positives, '-min' and '-range'.

The first parameter is applied to potential structural variations coming out from arrays 1a and 1b: basically, when '-min' is used, all potential structural variations shorter than one standard deviation are skipped. As previously stated, standard deviation is a way to mean the error that an arc can have; in general, the values in arrays 1a and 1b are linked to the length of indels causing them: the longer the indel, the higher the shifting of observed distribution, the bigger the values saved. Also the opposite is true: the shorter the indel, the smaller the shifting of observed distribution, the smaller the values saved. But values too small can come from a small indel as well as errors, randomness, etc, so sometimes it is very useful to skip over potential structural variations indicating too small indels, because it is more probable that they are false positive. By default, '-min' parameter is not used.

The second parameter is applied to potential structural variations coming out from array 4: basically, when '-range' is used, all potential structural variations longer than a certain value are skipped. This value depend from library average length, library standard deviation and '-direction' parameter:

- if '-direction' parameter is set, maximum length is $2 * (AvgLen + 4 * StdDev)$
- if '-direction' parameter is unset, maximum length is $2 * (2 * AvgLen + 7 * StdDev)$

Potential structural variations coming out from array 4 are regions with scattered single reads. In case of enough isolate insertions, the scattered region must stay within a certain range, i.e. the maximum length that an arc can have according to the library distribution, both on the left and on the right of the insertion point. If SV_finder can treat single reads as arcs (which means '-direction' parameter is set), the putative arcs can be at $AvgLen + 3 * StdDev$ bp of distance on both sides: one standard deviation as error is added, so the maximum length of $2 * (AvgLen + 4 * StdDev)$ is gotten. Otherwise, with '-direction' parameter unset, one single read give birth to one arc for each side, so maximum length is doubled: $2*(2*AvgLen+7*StdDev)$ (one standard deviation as error is added before the doubling for one side). By default, '-range' parameter is not used.

2.5 Second Steps

The second step is the feature that make SV_finder different from other tool for the prediction of structural variations. Basically, in the second step certain reads are considered and aligned onto the potential structural variations found during the first step (figure 2.10). This approach has several advantages:

- a predicted structural variation is backed up by two independent proofs
- potential structural variations from the first step are filtered during the second step by the spliced-alignment, cutting down the number of false positive
- structural variation breakpoints are computed with a base precision, no matter how great is the standard deviation of sequenced library
- by using only certain reads, the risk of obtaining non-sense splice-alignments is avoided
- as stated by other researcher[2], splice-alignments are a possible signature for structural variations; splice-aligning only on certain regions (the potential structural variations from the first step) makes it possible to use short reads, otherwise trying to splice-align sequence of 50 bp on the entire genome would result in a high number of false positive

A read that already align onto the reference genome without the need to splice is not considered; in order to consider a read for the splice-alignment, it has to not be aligned because between the reasons of the alignment failure can be present the fact that the read covers a breakpoint. This is only the first constraint: using only no-alignment as way to decide if a read can or cannot be splice-aligned, millions of reads are selected, and they are too much to assure as few as possible false positive. The second constraint comes from the sibling of a not-aligned read; there are two cases:

- sibling read is aligned; in this case, it decides the potential structural variations in which the not-aligned read can be splice-aligned (figure 2.12)
- sibling read is not aligned; in this case both reads are not aligned; there are many reasons for such behavior and, among them, there is the fact

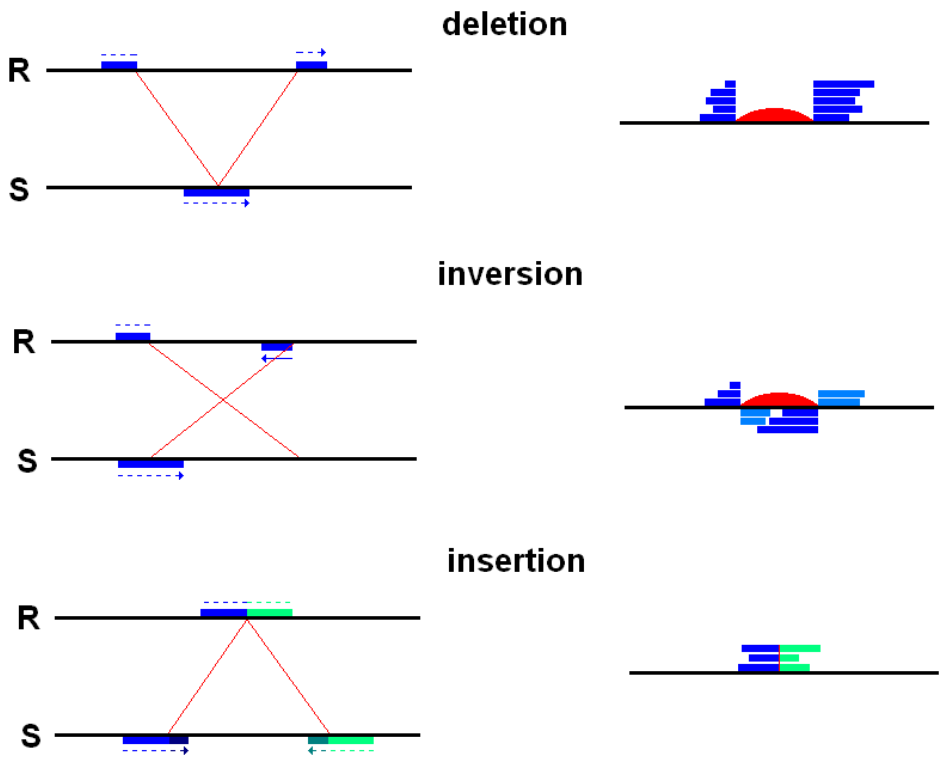

Figure 2.10: Schematic description of spliced-alignments identifying structural variations; on the left side it is showed how covering-breakpoint reads behave in case of deletion, inversion and insertion, with 'R' as the reference genome and 'S' as the query genome; on the right side it is showed how the spliced-alignment of those reads appears and point to the structural variation breakpoints.

that one read doesn't align because it covers a breakpoint, while its sibling doesn't align because it comes from an inserted sequence; so sibling not-aligned reads are splice-aligned only for insertions of the first step

Reads selected as above are spliced-align onto the potential structural variations from the first step. Because deletions and inversions interest an entire region (while insertions interest only a single point) of the reference genome, they are constrained; only the end regions of potential deletions and potential

inversions are used as reference to splice-align selected reads: assuming that each potential structural variations has at least one structural variation and knowing that within 3 standard deviations (from the average length) falls 99.6% of right arcs, the two breakpoints must are at 3 standard deviations from the end regions. To sum up (see figure 2.11):

- if a deletion or an inversion found after the first step is equal or shorter than $6*StdDev$, reads are splice-aligned over all the potential structural variation
- if a deletion or an inversion found after the first step is longer that $6 * StdDev$, reads are splice-aligned only on position p if $s \leq p \leq s+3*StdDev$ or $e-3*StdDev \leq p \leq e$, where s and e are respectively start position and end position of the potential structural variation

Figure 2.11: In case of deletions or inversions longer than $6 * StdDev$ (so $E - S > 6 * StdDev$), reads are splice-aligned only on red positions; if $E - S \leq 6 * StdDev$, reads are splice-aligned in all positions between S and E.

As previously stated, the decision to where splice-align a not-aligned read R depends from its aligned sibling S; if S align at position f_S (where $f_S = \frac{s_S+e_S}{2}$, with s_S as the start position and e_S as the end position where S aligns) and a potential structural variation PSV that interests position from the s_{PSV}-th to the e_{PSV}-th base is considered, R is splice-aligned onto PSV if:

- $s_{PSV} - 3 * StdDev \leq f_S \leq e_{PSV} + 3 * StdDev$, in case '-direction' parameter is unset
- in case '-direction' parameter is set, one the following conditions is true:
  - $s_{PSV} - 3 * StdDev \leq f_S < s_{PSV}$ and R lies on the right to S according to '-direction' parameter

- $s_{PSV} \leq f_S \leq e_{PSV}$
- $e_{PSV} < f_S \leq e_{PSV} + 3 * StdDev$ and R lies on the left to S according to '-direction' parameter

All the not-aligned reads that fulfill these conditions are splice-aligned onto PSV.

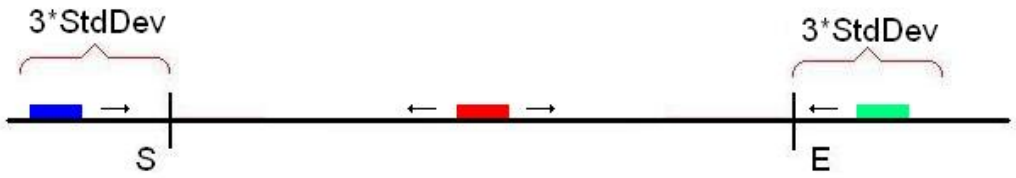

Figure 2.12: An aligned read can be in 3 different positions: on the left of region start within 3 standard deviation (blue read), inside region (red read) or on the right of region end within 3 standard deviation (green read). If no direction is specified, its not-aligned sibling is splice-aligned regardless the position; if direction is specified, blue read must have the sibling on the right and green read on the left.

2.5.1 Easy Breakpoints

The main point of the second step is the splice-alignment of those reads that could cover a breakpoint in order to detect with a base-precision the breakpoint itself. But how much is it easy to have such a read? A breakpoint is a point on a genome, so the question is: given a position i, what is the probability that a read R cover i? First of all, a read R cover i if, named s_R the start position and e_R the end position of an alignment of R, is true

$$s_R \leq i \leq e_R$$

Position i ideally cuts R into two pieces; for the purpose of splice-alignments, it is necessary that each piece is long enough to align with a good sensibility: if a piece is too short, it aligns everywhere, so it is useless. From a statistical point of view, a sequence of N nucleotides on average can be aligned one time every 4^N nucleotides so, for example, a piece of 4 nucleotides aligns one time every 256 nucleotides: a piece of such length obtains too many matches,

i.e. a lot of false positives. In order to avoid this, there must be a limit on the shortest length sl that a piece can have to consider it as aligned:

- for deletions and inversions, by default sl is set to 6 (one alignment every $4^6 = 4096$ nucleotides)
- for insertions, alignments of not-aligned reads anchored by their siblings, by default sl is set to 8 (one alignment every $4^8 = 65536$ nucleotides)
- for insertions, alignments of both not aligned sibling reads, by default sl is set to 10 (one alignment every $4^8 = 1048576$ nucleotides)

Minimum length sl is linked to the length of the reference against whom reads are splice-aligned and the number of reads itself. For deletions and inversions, the highest length of a reference is $6 * StdDev$; assuming a standard deviation of 200 bp, a maximum length of one thousand of nucleotides is gotten. Also reads for the splice-alignment are selected with a high sensibility, so a value of 6 for sl is a very good one. Insertions has a shorter reference compared to that for deletions and inversions, however one can assume a length of thousands nucleotides. The problem arises for reads to splice-align: a not-aligned reads anchored by its sibling can come out from two situations:

- the not-aligned read belongs to the inserted sequenced
- the not-aligned read covers one of the two breakpoints

When not-aligned reads are selected, both the above types of reads are taken: the first type contributes a lot to the number of reads to be splice-aligned, but they give at least false positive. Instead of having n reads to align (as happens for deletions and inversion), SV_finder has something like $100 * n$ reads. In order to avoid a lot of false positives, a higher sl is needed, so it is set about one order of magnitude higher than that for deletions and inversions. Last situation is the alignment of both not aligned sibling reads for insertions: in this case, there is no clue about the positions where reads can be splice-aligned, so they aligned over all the references for potential insertions. This means that the entire reference is one to two orders of magnitude higher than the normal situation for insertions, so the sl is much more higher.

It is easy to understand that the probability of a read covering a breakpoint is linked to the sequence coverage: a bigger sequence coverage means a

greater probability to have such reads, and viceversa. The sequence coverage follows a Poisson distribution, expressed by the following form:

$$f_{k;\lambda} = \frac{\lambda^k e^{-k}}{k!}$$

where

- $f_{k;\lambda}$ is the probability of an event occurring k times; in this case, it is the probability of having k reads covering a breakpoint
- k is the number of occurrences of an event; in this case, k is how many reads cover a breakpoint
- λ is the expected number of occurrences of the same event in the given interval; in this case, λ is the average sequence coverage

With a $1x$ sequence coverage, the probability p to have exactly one read that cover a breakpoint is:

$$p = \frac{1^1 e^{-1}}{1!} \approx 0.37$$

In order to get probability to have al least one read covering a breakpoint witn $1x$ sequence coverage, one must add the probabilities to have n reads covering the breakpoint, so the overall probability p is:

$$p = \sum_{k=1}^{\infty} \frac{1^k e^{-k}}{k!} \approx 0.44$$

A probability of 0.44 at $1x$ sequence coverage is very good: in about half of the structural variations, there is at least a read that cover the breakpoint, allowing SV_finder to efficiently use the second step.

However, it is important to consider sl: we are not interested to reads covering a breakpoint with their ends; in order to be useful, a read must cover a breakpoint with its middle positions. As result, the effective sequence coverage is lower that the real sequence coverage: int the previous example, the sequence coverage to use in the Possion expression is lower than $1x$. Under the hypothesis that a read is 50 bp long and $sl = 6$, the effective sequence coverage is $1 * (50 - 2 * 6)/50 = 0.76x$ so the real probability to have at least one read covering a breakpoint is:

$$p = \sum_{k=1}^{\infty} \frac{0.76^k e^{-k}}{k!} \approx 0.33$$

In about one third of all structural variations, there is at least one read covering a breakpoint. Simulations (see section 3.1) show a sensibility at each sequence coverage that well agrees to Poisson expression.

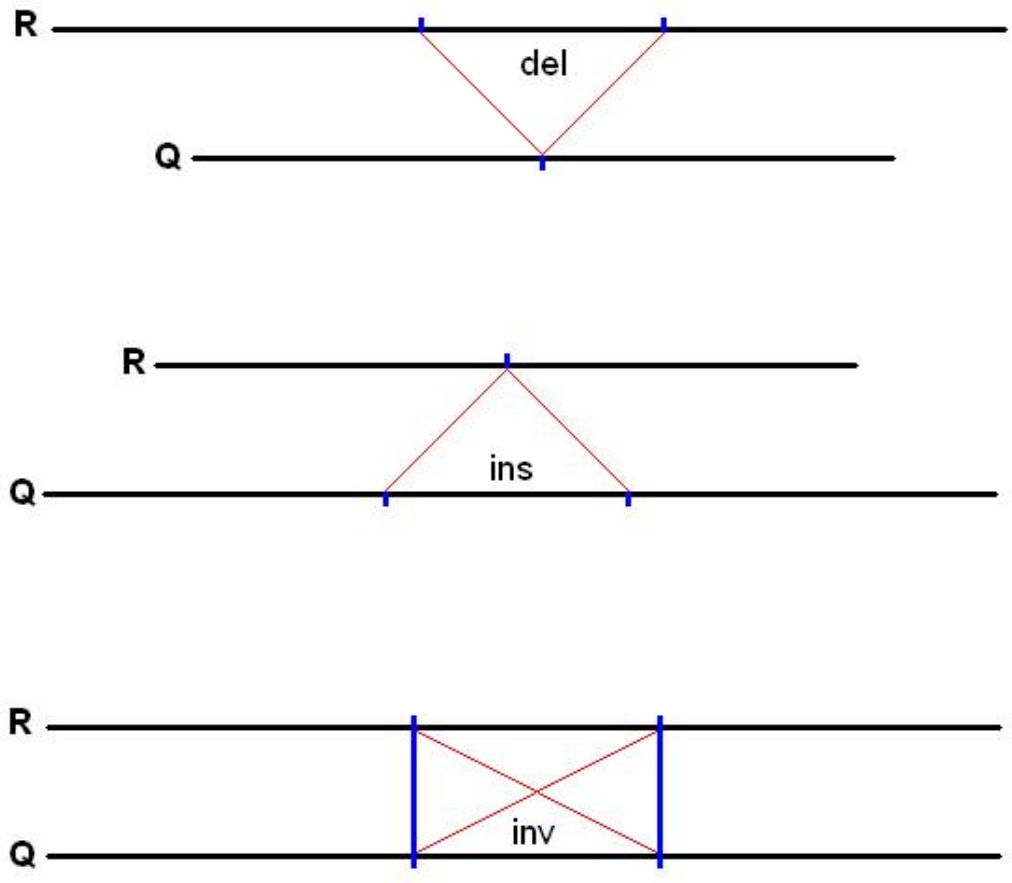

Figure 2.13: In blue, the breakpoints are marked: for deletions, one breakpoint on query genome and two on reference genome; for insertions, two breakpoints on query genome and one on reference genome; for inversions, two breakpoints both on query and reference genomes but, due to the reversion, each breakpoint point to the other.

It is also important to know how many breakpoints are needed to find a certain type of structural variation (see figure 2.13). A deletion is a region of DNA lacking on the query genome but present on the reference genome, so in the query genome we have only one breakpoint (that is the deletion point) and on the reference genome we have two breakpoints, the start position and the end position of the deletion. A read that covers the only breakpoint on the query genome splits into two pieces when aligned against the reference, each piece marking one of the two breakpoints, so for deletions it is needed at least one read covering the only one breakpoint in order to detect the deletion itself.

An inversion is a region of DNA present both on the query and reference genomes, so there are two breakpoints in both genomes. In order to detect an inversion, it is enough that only one of the two breakpoints is covered on the query genome, because its splice-alignment always detect both breakpoints on the reference genome, independently from which breakpoint on the query genome is covered: this only changes how the splice-alignment must be translated into an inversion. So inversions are found with the same probability of deletions, in regard of having reads covering breakpoints.

An insertion is a region of DNA present on the query genome but lacking on the reference genome, so it has two breakpoints on the query genome and only one on the reference genome. A read covering the breakpoint is aligned on the reference only for the piece outside the insertion, not the inside one; as result, a read covering the one of the two breakpoints marks the insertion only breakpoint on the reference but cannot detect it with a base precision. Moreover, the splice-alignment is only for pieces shorter than the read itself, so the number of matches between nucleotides are fewer, so the overall strength of alignment is weaker than those alignments that detect deletions and inversions (for which the entire read must be splice-aligned, not only a piece of it). This could be result in too much false positives. In order to avoid them, to detect insertions with base precision and the same strength for alignments, it is needed that both the breakpoints on the query genome are covered by at least one read each, so it's more difficult to detect insertions. In the previous example, at $1x$ sequence coverage deletions and inversions can be found with a sensibility[9] of 0.33 because only one breakpoint must be covered; because two breakpoints must be covered, insertions can

[9]the probability to have a read covering a breakpoint is very close to the sensibility in detecting structural variations

be found with a sensibility of $0.33^2 \approx 0.11$.

2.5.2 Parameters

The second step is all about an alignment, so there are a lot of parameters to define how the alignments must be done by SV_finder. With the exception of one, all these parameters are computed by default and are designed to obtain the best possible result with alignments of sequences 50 bp length. Roughly, parameters can divided into three classes:

- parameters for the basic alignment
- parameters for the anchoring of reads
- parameters for insertions

Parameters for the basic alignment basically tells the program how many mismatches can be allowed to any alignment; mismatches are divided into two types: general mismatches within the sequence and mismatches at the sequence ends (of pieces aligned). This distinction is important because the sequence ends are involved into detection of breakpoints and speed of alignment; a read that split into two pieces has 4 sequence ends:

- the outer ends are linked to the speed of program in doing the alignments; SV_finder takes the reference and slides over it all the reads selected for that reference so, whereas already the firsts nucleotides are different, SV_finder knows that an alignment cannot be done and skip to the next position on the reference. Allowing too much mismatches at the sequence ends means slowing down too much the program
- the inner ends detect the breakpoints; sometimes there can be an error (or other problems, such as SNP) on the nucleotides that detect breakpoints, so mismatches must be allowed; in case of color space, regularly there are some sort of shifting between alignment and breakpoint, so one or more mismatches are a must

General mismatches within the sequence are needed mostly to take into account the presence of sequencing errors and SNP, in order to allow the alignment of a read that does not perfectly match with the reference due to them. There is also another parameter for mismatches within the sequence that is

needed to control alignment of pieces very short regard the read itself: it was discussed in section 2.2.6.

As stated in section 2.5.1, a read must anchor with a minimum number of nucleotides to be considered as aligned; this anchoring is basically the *sl* explained there. As stated, three different values of *sl* are needed, one for deletions and inversions and two for the two different types of reads used for insertions.

Insertions are very different from deletions and inversions, because the read splice-alignment doesn't immediately give the breakpoint of the insertion. Also only a piece of a read is aligned, not the entire read, so false positive are common. In order to avoid them, only the longest alignment are kept, so a parameter is needed to tell SV_finder how long an alignment must be to keep it. When there are two alignments from two different reads[10] that point the same position with different side of the reads (figure2.14), the insertion can be claimed, but:

- the two alignments can overlap, so the program must know the maximum allowed overlapped
- because probability that an alignment is found by change depends from number of matches, to equal probabilities between insertions, deletions and inversions, a minimum number of matches for the two reads detecting the insertion is needed; so SV_finder wants a minimum number for the sum of the lengths of the two reads that see each other

2.5.3 Spliced Alignment

For every structural variation, the first step is selecting the read to splice-align and getting the reference against which align them; both these process is described in section 2.5. Then the main point is the splice-alignment, that is done with a simple sliding of each read over the reference using the parameters described in section 2.5.2. But also the management of the alignments plays a central role: it is necessary to translate splice-alignments into structural variations, and this is not as simple as it sounds.

[10] such two reads are 'seeing each other'

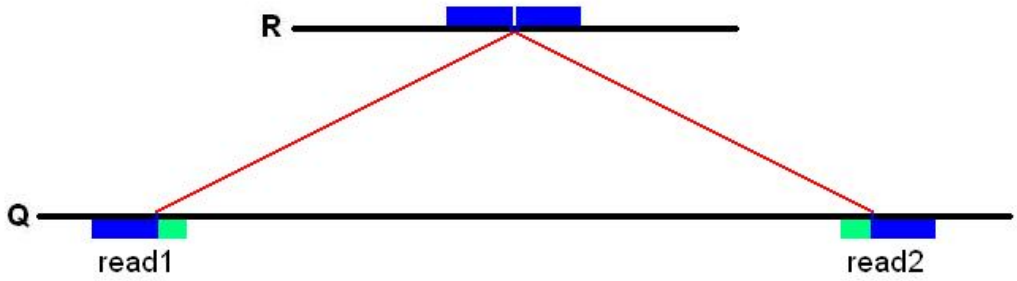

Figure 2.14: In an insertion, read1 and read2 cover the two breakpoints on the query genome; when the reads are aligned against the reference, only the blue pieces of each read (the ones that are outside the insertion) align and they are placed so each one is where the green piece of the other read should be: the two pieces 'see each other'.

Deletion Spliced Alignment

Deletions have the most basic work-flow, mainly because the idea of the splice-alignment is drawn from the transcriptome analysis for splicing site: introns cut away from the immature RNA to form the mature RNA is a situation very close to deletions. The steps to translate splice-alignments into deletions are the following:

1. a read R is aligned against the reference in 4 ways to obtain its pieces aligned:
   - from the start point (piece S of length l_S)
   - from the end point (piece E of length l_E)
   - from the start of reverse complement (piece S^{-1} of length $l_{S^{-1}}$)
   - from the end of reverse complement (piece E^{-1} of length $l_{E^{-1}}$)
2. all the pieces are combined against each other; if l_R is the length of R, a combination is found when one of the following conditions is met:
   - $l_S + l_E > l_R$
   - $l_S + l_E = l_R$
   - $l_R - 2 * M_{IntMis} \leq l_S + l_E < l_R$ with M_{IntMis} is the maximum number of allowed end mismatches

- $l_{S^{-1}} + l_{E^{-1}} > l_R$
- $l_{S^{-1}} + l_{E^{-1}} = l_R$
- $l_R - 2 * M_{IntMis} \leq l_{S^{-1}} + l_{E^{-1}} < l_R$ with M_{IntMis} is the maximum number of allowed end mismatches

3. all the above combinations form deletions; start and end of the deletion is computed, according to the combination itself

4. if a read has no combinations, its longest piece alignments are saved

5. after each read is searched for combinations, longest piece alignments of reads without combinations are clustered with combinations of other reads

6. combinations supported by the highest number of reads are clustered together and saved as deletion for the reference analyzed

Insertion Spliced Alignment

Insertions have a particular work-flow, mainly due to the fact that the splice-alignments need to be refined in order to get the insertions. Also two types of reads are splice-aligned: not_aligned reads with siblings aligned and both not-aligned sibling reads. The steps to translate splice-alignments into insertions are the following:

1. a read R is aligned against the reference in 4 ways to obtain its pieces aligned:
   - from the start point (piece S of length l_S)
   - from the end point (piece E of length l_E)
   - from the start of reverse complement (piece S^{-1} of length $l_{S^{-1}}$)
   - from the end of reverse complement (piece E^{-1} of length $l_{E^{-1}}$)

2. longest alignments are saved
   - not-aligned reads with siblings aligned are aligned against one insertion reference, so longest pieces against that reference are saved

- both not-aligned sibling reads are aligned against all insertion references and the longest pieces among all are saved

3. longest alignments for all involved reads are collected
4. alignments of start pieces (S and S^{-1}) are put against alignment of end pieces (E and E^{-1})
5. an insertion is found when:
   - at least a start piece and an end piece see each other
   - the sum of the two pieces are longer that the average length of a read
6. breakpoint of the insertion is found according to aligned pieces

Inversion Spliced Alignment

Inversions has a work-flow similar to that for deletions, but with one big difference: the splice-alignment interests opposite strands; also there are four different ways for a read to cover the breakpoints and the way chosen must be recognized (see figure 2.15). The steps to translate splice-alignments into inversions are the following:

1. a read R is aligned against the reference in 4 ways to obtain its pieces aligned:
   - from the start point (piece S of length l_S)
   - from the end point (piece E of length l_E)
   - from the start of reverse complement (piece S^{-1} of length $l_{S^{-1}}$)
   - from the end of reverse complement (piece E^{-1} of length $l_{E^{-1}}$)
2. all the pieces are combined against each other; if l_R is the length of R, a combination is found when one of the following conditions is met:
   - $l_S + l_{S^{-1}} > l_R$
   - $l_S + l_{S^{-1}} = l_R$
   - $l_R - 2 * M_{IntMis} \leq l_S + l_{S^{-1}} < l_R$ with M_{IntMis} is the maximum number of allowed end mismatches

- $l_S + l_{E^{-1}} > l_R$
- $l_S + l_{E^{-1}} = l_R$
- $l_R - 2 * M_{IntMis} \leq l_S + l_{E^{-1}} < l_R$ with M_{IntMis} is the maximum number of allowed end mismatches

3. all the above combinations form inversions; start and end of the inversion is computed, according to the combination itself and based on how a read cover the breakpoint

4. if a read has no combinations, its longest piece alignments are saved

5. after each read is searched for combinations, longest piece alignments of reads without combinations are clustered with combinations of other reads

6. combinations supported by the highest number of reads are clustered together and saved as inversion for the reference analyzed

2.5.4 Color Space

SOLiD sequencing introduced a new method for obtaining nucleotide sequences: instead of giving a sequence of A, C, G and T, commonly referred as 'base space', its output is made up of sequences of 0, 1, 2 and 3, named 'color space'[11]. A color represents two adjacent nucleotides:

- DNA is made of 4 nucleotides
- taking 2 nucleotides as once gives birth to 16 different combinations ($4^2 = 16$)
- SOLiD use only 4 colors for all combinations

The above considerations mean that a color could translate in 4 different couples of bases as in picture 2.16:

In order to have a unique translation of colors into bases, for each color at least one base must be know. A sequence of colors can be translated into 4 different base sequences, so if one nucleotide is known a unique translation is

[11] on SOLiD manual, the output is showed with 4 different colors, but in real life it is showed with numbers

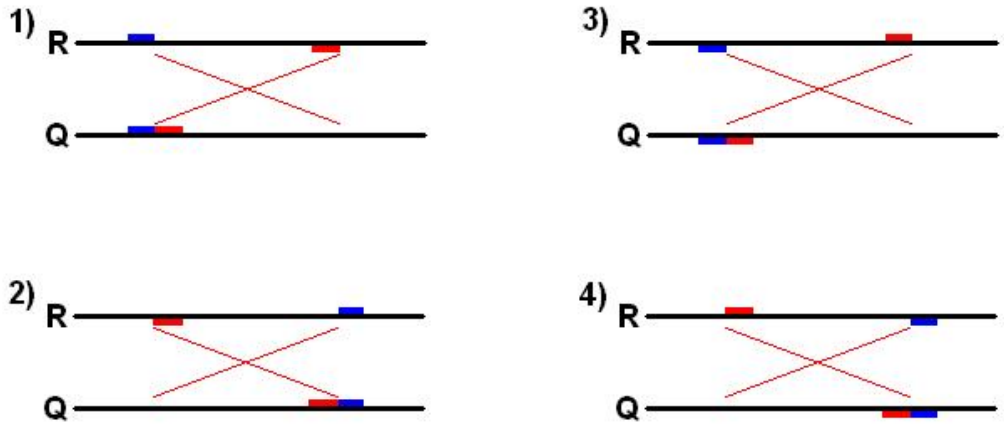

Figure 2.15: An inversion and a read covering a breakpoint are showed; read is divided into two pieces: blue one is the piece outside the inversion, red one is inside. Blue piece aligns always in the same breakpoint and strand against the reference, while red piece aligns on the other breakpoint and strand: 1) read covers the first breakpoint on '+' strand; 2) read covers the first breakpoint on '-' strand; 3) read covers the second breakpoint on '+' strand; 4) read covers the second breakpoint on '-' strand.

made. In general, the first nucleotide is always a 'T', so the following colors can have a unique translation:

- the first color depends from the first and second nucleotide
- the first nucleotide is the 'T' base
- knowing the first nucleotide makes the second nucleotide known as well
- the second color depends from the second and third nucleotide
- the second nucleotide is now known
- so the third nucleotide is now determined
- and so on, until the last color

In order to determine the i-th base, the $i-1$-th color and the $i-1$-th base must be known.

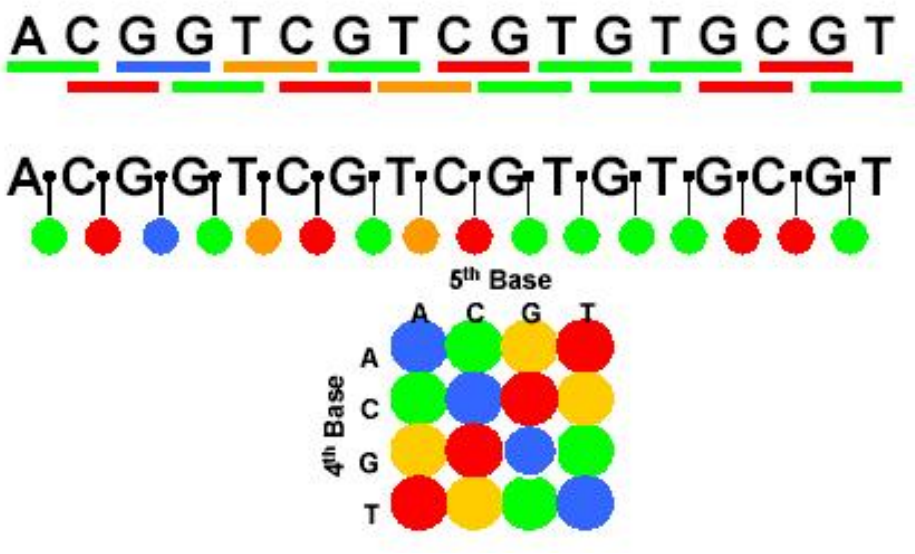

Figure 2.16: SOLiD encoding: 4 colors encode for 16 different di-base.

A color space sequence can be translated into a base space sequence by knowing only one nucleotide over the sequence (in general, it is the first one, the 'T' nucleotide, as previously stated), but in practice the color sequence in first aligned against a reference and then translated. The reason is very simple: all the nucleotide following the i-th position depend from the $i-1$-th color; if this color is wrong, all those nucleotides are wrong. In case of sequencing error, translating the sequence before aligning it against a reference could lead to completely wrong sequences. If the sequence is aligned and then is translated, errors can be detected, as well as SNPs: in case of a SNP, one base changes, so two colors change in a color sequence and one sees two mismatches in the sequence alignment; if only one color changes, only one mismatch is seen, so it is a sequencing error.

The SOLiD color space schema is specifically designed to have some special properties:

- for each couple of bases, the reverse is always the same color
- for each couple of bases, the complement is always the same color
- for each couple of bases, the reverse complement is always the same

color

As stated by Applied BioSystem, this schema enables the unique error checking capability of the SOLiD System.

2.5.5 Breakpoint Detection

A major point of SV_finder is the breakpoint detection with a base precision, allowed by using spliced-alignments. For alignments in base space, it is quite simple to translate spliced-alignment into precise breakpoints. For color space, the things change badly: the color that depends from the outer and inner nucleotides (figure 2.17) respect to the structural variation is different from the color one expect, and the difference is semi-random, so color space adds a greater error on breakpoint detection. SV_finder makes into accounts all the potential situations in order to have the minimum possible error and not mistake the right breakpoint.

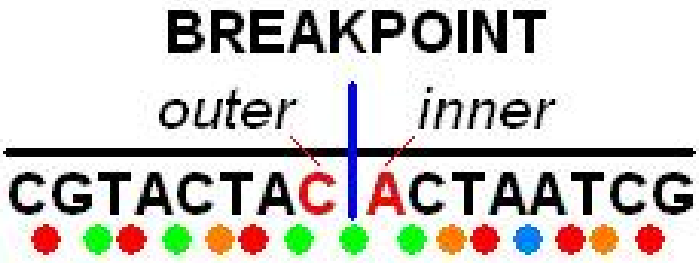

Figure 2.17: Blue line is the breakpoint: red nucleotides are the outer and inner ones.

Deletion

When a deletion is present, there are one breakpoint on the query genome and two breakpoints on the reference genomes, because the start and end of the deletion are adjacent on the query. Only one sequence that covers the breakpoint is needed to locate deletion start and end (figure 2.18): this sequence splice-aligns on the reference genome into two different pieces that identify 4 positions; the most inner positions are the start and the end that SV_finder is looking for.

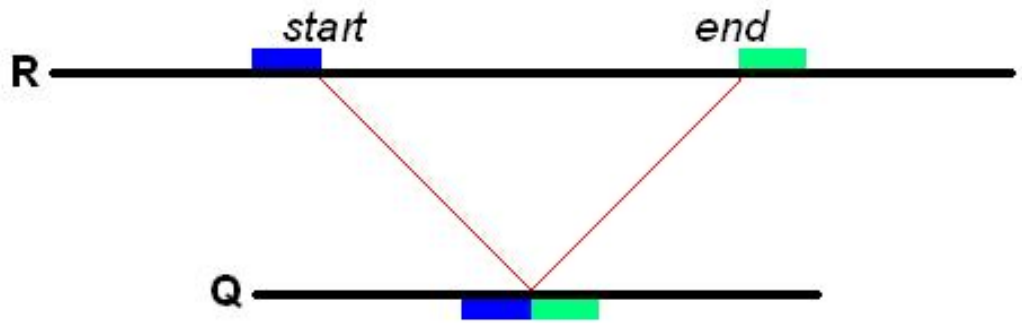

Figure 2.18: The read showed (in blue e green color) covers the breakpoint for the deletion; one aligned against the reference, blue piece marks the deletion start and green piece marks the deletion end: only one read is needed to find the deletion.

The previous situation is an ideal one: it's unlikely that a sequence covering the breakpoint splits perfectly to locate with absolute precision start and end. Often, due to errors, SNPs or simply randomness, start and end have some degree of uncertainty. For deletion, a sequence covering the breakpoint splits into two pieces that align in positions i, j, k and l respectively, with i and j start and end of the first piece, k and l start and end of the second piece and $i < j < k < l$. Deletion start and end depend from j and k respectively.

The following examples explain the problem of computing deletion start and end from j and k.

$$R_{gen}: \quad AAACGACTTGTTGT|tagcga|CACACGTCTACGCT$$

$$R_{read}: \quad TTGT| \quad |CACA$$

$$Q_{gen}: \quad AAACGACTTGTTGT|CACACGTCTACGCT$$

$$Q_{read}: \quad TTGT|CACA$$

in the above schema:

- lines with a 'R' in the beginning refers to the reference genome

- lines with a 'Q' in the beginning refers to the query genome
- first and third lines are the genomes
- second and forth lines are a read covering the breakpoint how as it appears aligned against its genome (the query one) and the reference
- the | symbol marks a breakpoint
- the sequence *tagcga* in lower-case is the deletion and, obviously, is not present on the query

The situation described above is an ideal one: the sequence splits perfectly into 2 pieced of 4 bp, and the alignment coordinates of forth (j) and fifth (k) bases locate deletion start S_D and end E_D, respectively: $S_D = j + 1$ and $E_D = k - 1$, both with an error of 0.

Let's change only one nucleotide and analyse the new situation:

$$\begin{aligned} R_{gen}: &\quad AAACGACTTGTTGT|tagcg\underline{t}|CACACGTCTACGCT \\ R_{read1}: &\quad\quad\quad\quad\quad\quad TTGT| \quad\quad |CACA \\ R_{read2}: &\quad\quad\quad\quad\quad\quad TTG \;\; | \quad\quad t|CACA \end{aligned}$$

$$\begin{aligned} Q_{gen}: &\quad AAACGACTTGTTGT|CACACGTCTACGCT \\ Q_{read}: &\quad\quad\quad\quad\quad\quad TTGT|CACA \end{aligned}$$

Only the last base of deletion is changed (it is underlined), so its new sequence is *tagcgt*. Lines beginning with R_{read1} and R_{read2} show the two possible alignments for the read covering the breakpoint: first alignment is the same in the first situation and correctly identify start and end; but second alignment is a bit different and identify two shifted coordinates for the deletion. This happens by chance: there is a 0.25 probability that nucleotides outer the breakpoint match with nucleotide inner the breakpoint, so start and end shift of one position. More than one nucleotide can match: in case of two, the probability of such an event is 0.0625 and the corresponding shifting is of two positions, and son on.

It is important to note that the strength of an alignment is directly proportional to the number of matches: more equal nucleotides between query

and sequence means a stronger alignment. Based on this idea, SV_finder splice-aligns the reads that could cover a breakpoint and search for the longest piece alignment available. For this reason, in the above example the two pieces found are the first one from R_{read1}, so piece $TTGT$, and second one from R_{read2}, so $tCACA$. The t is in a ambiguous position: does it belong to the second piece? Or to the first one? SV_finder cannot know, so if

- the first piece has a length l_{p1}
- the second piece has a length l_{p2}
- the read has a length of l_R
- the overlap ol between the two pieces is $ol = l_{p1} + l_{p2} - l_R$

the position of that t in both pieces is the j and k coordinates needed, and deletion start S_D and end E_D are $S_D = j + 1 - ol/2$ and $E_D = k - 1 + ol/2$, both with an error of $ol/2$.

A third situation is possible:

$$R_{gen}: \quad AAACGACTTGTTGT|tagcga|CACACGTCTACGCT$$

$$R_{read}: \quad TTGT| \qquad |gACA$$

$$Q_{gen}: \quad AAACGACTTGTTGT|GACACGTCTACGCT$$

$$Q_{read}: \quad TTGT|GACA$$

In the query genome, the first base after the breakpoint is changed, due to sequencing error or a SNP (it doesn't matter). Now the alignment of the second piece is different: the first base, a G, doesn't match anymore with the C present in the reference, so now the deletion end that SV_finder computes is shifted one position on the right of the real deletion end. Where do I put that g? On the first piece? On the second one? Maybe must it be not considered? Again, SV_finder cannot know the answers so, if we are in the same condition as above, with the only exception that there is no overlap but a certain number num_M on mismatches of $num_M = l_R - l_{p1} - l_{p2}$, it happens that S_D and E_D are $S_D = j + 1 + num_M/2$ and $E_D = k - 1 - num_M/2$, both with an error of $num_M/2$. More than one nucleotide can mismatch, for a greater error.

Insertion

When an insertion is present, there is the opposite situation of a deletion: two breakpoints are on the query genome, and only one breakpoint on the reference; start and end of an insertion are adjacent in the query, while in the reference from a unique point (for this reason, insertion has a start but not an end, see section 2.7). Unfortunately, both the breakpoints in the query must be covered by two different reads in order to detect the insertion, so finding insertions is more difficult compared to finding deletions and inversions. A read that cover one of the two query breakpoint has a piece inside the insertion and a piece outside; only the latter align against the reference: the piece inside an insertion is not present on the reference genome[12]. This means that two reads covering the two breakpoints align on the reference as follow:

$$
\begin{aligned}
R_{gen}: &\quad CGACTTGTTGT|CACACGTCTAC \\
R_{readA}: &\quad \underline{TTGT}| \\
R_{readB}: &\quad |\underline{CACA}
\end{aligned}
$$

$$
\begin{aligned}
Q_{gen}: &\quad CGACTTGTTGT|tagcgatgctg|CACACGTCTAC \\
Q_{readA}: &\quad \underline{TTGT}|tagc\quad| \\
Q_{readB}: &\quad |\quad gctg|\underline{CACA}
\end{aligned}
$$

with:

- *tagcga* is the inserted sequence and is present only in the query genome
- Q_{readA} is the read covering the first breakpoint
- Q_{readB} is the read covering the second breakpoint
- R_{readA} is the alignment of the read covering the first breakpoint

[12] in case of insertion of novel sequence; if a sequence present elsewhere is inserted, then is very unlikely that it is within the region where the splice-alignment is done, so in both case the piece inside the insertion cannot align in the region found to be potential interested by an insertion during the first step

- R_{readB} is the alignment of the read covering the second breakpoint

Only the underlined pieces of both reads align against the reference: the piece aligned 'see each other':

$$\underline{TTGT}|\underline{CACA}$$

in this way they detect the insertion breakpoint, because the last nucleotide of the first piece identifies the nucleotide before the breakpoint, while the first nucleotide of the second piece identifies the nucleotide after the breakpoint. There are two pieces aligned in positions i, j, k and l respectively, with i and j start and end of the first piece, k and l start and end of the second piece, as happens for deletions: the only difference is that the pieces come from different reads, not the same one. In such a situation, the insertion start S_I is $S_I = j = k - 1$[13] with no error; there are no end.

The previous situation is an ideal one: it's unlikely that two pieces see each other so perfectly. Very often, due to errors, SNPS ann randomness, the two pieces overlap, even for a considerable number of positions, or are not adjacent. Let's change only two nucleotides and analyse the new situation:

$$\begin{aligned}
R_{gen} &: \quad CGACTTGTTGT|CACACGTCTAC \\
R_{readA} &: \quad \underline{TTGT}| \\
R_{readB1} &: \quad |\underline{CACA} \\
R_{readB2} &: \quad \mathbf{T}|\underline{CACA} \\
R_{readB3} &: \quad \mathbf{GT}|\underline{CACA}
\end{aligned}$$

$$\begin{aligned}
Q_{gen} &: \quad CGACTTGTTGT|tagcgatgc\mathbf{gt}|CACACGTCTAC \\
Q_{readA} &: \quad \underline{TTGT}|tagc \quad | \\
Q_{readB} &: \quad | \quad gc\mathbf{gt}|\underline{CACA}
\end{aligned}$$

The last two nucleotides of the inserted sequence (in bold) are changed in a way that they match with the first two nucleotides before the insertion breakpoint on the reference. Now $readB$ has 3 way to align: R_{readB1} that

[13] if the insertion point is p, it means that the inserted sequence starts from position $p + 1$

detects the correct breakpoint, R_{readB2} that is shifted one base to the left and R_{readB3} that is shifted two bases to the left. As for deletions, SV_finder splice-aligns the reads that could cover a breakpoint and search for the longest piece alignment available: only R_{readB3} is used to find the insertion. It overlaps with the alignment of the first read:

$$R_{readA}: \qquad \underline{TTGT}|$$

$$R_{readB3}: \qquad \mathbf{GT}|\underline{CACA}$$

Now let's change only one nucleotide and analyse the new situation again:

$$R_{gen}: \qquad CGACTTGTTGT|CACACGTCTAC$$

$$R_{readA}: \qquad \underline{TTGT}|$$

$$R_{readB}: \qquad\qquad\quad |\ \underline{ACA}$$

$$Q_{gen}: \qquad CGACTTGTTGT|tagcgatgctg|\mathbf{G}ACACGTCTAC$$

$$Q_{readA}: \qquad \underline{TTGT}|tagc\qquad |$$

$$Q_{readB}: \qquad\qquad |\qquad gctg|\underline{\mathbf{G}ACA}$$

In this case, the first nucleotide after the second breakpoint (in bold) is changed from a C to a G. The alignment of the second piece is shorter, only 3 matches, against the 4 that would be if no difference took place.

In the last two examples, overlap and pieces not adjacent, the solution is the same: start is $S_I = \frac{k-1+i}{2}$ and its error is $E_S = \frac{|k-1-1|}{2}$.

Inversion

When an inversion is present, there are two breakpoints in both query and reference genome, because no DNA piece is lacking or added but a certain region in reversed. If a sequence covers one of the two breakpoints on the query, its spliced-alignments on the reference locate both inversion start and end: the difference between having a read covering the first breakpoint or having a read covering the second one lies only on how spliced-alignments coordinates are translated into inversion start and end (see figure 2.15 on section 2.5.3).

As for deletions, the situation described in figure 2.15 is an ideal one for the same reasons. A sequence covering one of the two breakpoints splits into two pieces that align in positions i, j, k and l respectively, with i and j start and end of the first piece, k and l start and end of the second piece and $i < j < k < l$. Inversion start depends from i or j, while inversion end depends from and k or l.

The following examples explain the problem of computing inversion start and end. For easiness, I consider read covering a breakpoint only from the same strand of the reference, but in case of reads from the opposite strand it's enough to compute reverse complement of the read to recreate the same situation.

$$\begin{array}{ll} R_{gen}: & AAACGACTTGTTGT|tagcgatccgat|CACACGTCTACGCT \\ R_{read}: & \qquad\qquad TTGT| \qquad\quad cgat| \end{array}$$

$$\begin{array}{ll} Q_{gen}: & AAACGACTTGTTGT|atcggatcgcta|CACACGTCTACGCT \\ Q_{read}: & \qquad\qquad TTGT|atcg \qquad\quad | \end{array}$$

in the above schema:

- lines with a 'R' in the beginning refers to the reference genome
- lines with a 'Q' in the beginning refers to the query genome
- first and third lines are the genomes
- second and forth lines are a read covering the **first breakpoint** how as it appears aligned against its genome (the query one) and the reference
- the | symbol marks a breakpoint
- the sequence *tagcgatccgat* in lower-case is the inversion and, obviously, is present on the query as a reverse complement, so *atcggatcgcta*

The situation described above is an ideal one: the sequence splits perfectly into 2 pieces of 4 bp, and the alignment coordinates of forth (j) and fifth[14] (l) bases locate inversion start S_V and end E_V, respectively: $S_V = j + 1$ and $E_V = l$, both with an error of 0.

If the read cover the second breakpoint, the situation is the following:

$$R_{gen}: \quad AAACGACTTGTTGT|tagcgatccgat|CACACGTCTACGCT$$

$$R_{read}: \quad |tagc \qquad |CACA$$

$$Q_{gen}: \quad AAACGACTTGTTGT|atcggatcgcta|CACACGTCTACGCT$$

$$Q_{read}: \quad | \qquad gcta|CACA$$

As usual, we are interested to forth and fifth bases, but now they have alignment coordinates of i and k respectively, and so insertion start and end are $S_V = i$ and $E_V = k - 1$, both with an error of 0.

As for deletions, the previous are ideal situations; let's change only one nucleotide and see what happens:

$$R_{gen}: \quad AAACGACTTGTTG\underline{G}|tagcgatccgat|CACACGTCTACGCT$$

$$R_{read1}: \quad TTGG| \qquad cgat|$$

$$R_{read2}: \quad TTG \ \ | \qquad cgat|C$$

$$Q_{gen}: \quad AAACGACTTGTTG\underline{G}|atcggatcgcta|CACACGTCTACGCT$$

$$Q_{read}: \quad TTGG|atcg \qquad |$$

Only the last base before the first breakpoint is changed (it is underlined), from a T to a G. Lines beginning with R_{read1} and R_{read2} show the two possible alignments for the read covering the first breakpoint: first alignment is the same in the previous situation and correctly identify start and end; but

[14] the second piece is *atcg*; when I want to align it I must compute its reverse complement, *cgat*; in the reverse complement, the last nucleotide is the first of the original piece so, in the example, the fifth nucleotide in the original read is the last nucleotide that one see in the alignment of the second piece

second alignment is a bit different and identify shifted coordinates for the inversion. As for deletions, this happens by chance and more nucleotides can match for greater than one shifts.

As for deletions, SV_finder splice-aligns the reads that could cover a breakpoint and search for the longest piece alignment available. For this reason, in the above example the two pieces found are the first one from R_{read1}, so piece $TTGT$, and second one from R_{read2}, so $cgatC$. The G (in the first piece; the second is a reverse complement, so the G becomes a C) is in a ambiguous position that SV_finder cannot know, so if

- the first piece has a length l_{p1}
- the second piece has a length l_{p2}
- the read has a length of l_R
- the overlap ol between the two pieces is $ol = l_{p1} + l_{p2} - l_R$

the position of that G in both pieces is the j and l coordinates needed, and inversion start S_V and end E_V are $S_V = j + 1 - ol/2$ and $E_V = l - ol/2$, both with an error of $ol/2$.

What happens if the read covers the second breakpoint? Basically, the same thing:

$$
\begin{array}{lcccc}
R_{gen}: & AAACGACTTGTTG\underline{G}|tagcgatccgat & | & CACACGTCTACGCT \\
R_{read1}: & |tagc & | & CACA \\
R_{read2}: & G|tagc & | & \ ACA
\end{array}
$$

$$
\begin{array}{lcccc}
Q_{gen}: & AAACGACTTGTTG\underline{G}|atcggatcgcta & | & CACACGTCTACGCT \\
Q_{read}: & | & gcta & |CACA
\end{array}
$$

As usual, SV_finder takes the longest alignment for each piece and combine them; as in the previous example, the position of G[15] in both pieces is the i and k coordinates needed, and inversion start S_V and end E_V are $S_V = i + ol/2$ and $E_V = k - 1 + ol/2$, both with an error of $ol/2$.

[15] i.e. the fifth nucleotide on the original read who remains a C in the second piece of R_{read1} and becomes a G in the first piece of R_{read2}

A third situation is possible:

R_{gen} : $AAACGACTTGTTGT|tagcgatccgat|CACACGTCTACGCT$

R_{read} : $TTG \;|\; cgat|$

Q_{gen} : $AAACGACTTGTTG\underline{C}|atcggatcgcta|CACACGTCTACGCT$

Q_{read} : $TTGC|atcg \;|$

In the query genome, the last base before the first breakpoint is changed, due to sequencing error or a SNP (it doesn't matter), from a T to a C. Now the alignment of the first piece is different: the last base, a C, doesn't match anymore with the T present in the reference, so now the inversion start that SV_finder computes is shifted one position on the left of the real inversion start_error. The situation si very close to what happens for deletions: SV_-finder cannot know where the mismatching nucleotide must be put so, if we are in the same condition as above, with the only exception that there is no overlap but a certain number num_M on mismatches of $num_M = l_R - l_{p1} - l_{p2}$, it happens that S_V and E_V are $S_V = j + 1 + num_M/2$ and $E_V = l + num_M/2$, both with an error of $num_M/2$. More than one nucleotide can mismatch, for a greater error.

If the read covers the second breakpoint, the error on the first doesn't matter; but an error on the second changes things:

R_{gen} : $AAACGACTTGTTGT|tagcgatccgat|CACACGTCTACGCT$

R_{read} : $|tagc \;|\; ACA$

Q_{gen} : $AAACGACTTGTTGT|atcggatcgcta|\underline{T}ACACGTCTACGCT$

Q_{read} : $|\; gcta|TACA$

Now the first base after the second nucleotide is changed, from a C to a T. As in the previous situation, the mismatching nucleotide (the T, fifth nucleotide of the original read) hasn't a defined position, so S_V and E_V are $S_V = i - num_M/2$ and $E_V = k - 1 - num_M/2$, both with an error of $num_M/2$

A final observation: it is possible that a sequencing error or a SNP change the things in way that a shifted perfect spliced-alignments is found, thus detecting the inversion one or more more positions shifted from the real ones. The following exmpla show this:

$$R_{gen}: \quad AAACGACTTGTTGT|tagcgatccgat|CACACGTCTACGCT$$

$$R_{read}: \quad TTG \;|\; cgat|C$$

$$Q_{gen}: \quad AAACGACTTGTTG\underline{G}|atcggatcgcta|CACACGTCTACGCT$$

$$Q_{read}: \quad TTGG|atcg \;|$$

The splicing is perfect: no overlap, no mismatch; but the inversion start and end found are wrong! They are shifted one base on the left and on the right, respectively, so the inversion is 2 bases larger and error on positions is 0. This situation happens by changes, and SV_finder cannot solve it. The only solution is using more relaxed parameters: inversions will be found with a greater error (only of few bases in any case), but noone will be missed.

2.5.6 Color Space Makes Things Difficult

The color space makes the breakpoint detection more difficult. The problem lies on the double-encoding: a color depends from two adjacent bases. But what happens in these bases are the ones before and after a breakpoint?

$$C|A$$

$$1$$

In the example above, the pipe locate a breakpoint with a C on the left and an A on the right; in color space, the di-base CA is translated into a 1[16]. If this breakpoint is the first one, nucleotide C is outside the structural variations and A is inside, so A will be replaced by another nucleotide, depending on the type on structural variation. The consequence is clear: if A changes, di-base CA changes and color 1 changes. Then there is a read covering this breakpoint: how does this read behave? The change of the color is

[16] as its reverse, AC, and their complement, GT and TG

semi-random: basically, it depends from the type of structural variation and from the nucleotide that occupy a certain position. Some examples help to explain this idea.

$$R_{gen}: \quad AAACGACTTGTTGT|\underline{t}agcga|CACACGTCTACGCT$$

$$Q_{gen}: \quad AAACGACTTGTTG\mathbf{T}|\underline{C}ACACGTCTACGCT$$

Here there are a deletion (sequence *tagcga* in lower case); the two underlined nucleotides must be the same in order to have the same color in the sequence or, at least, a sequencing error or a SNP must change the bold nucleotide to recreate a di-base translated into the same color, for example a C[17]. In the last (less probable) case, the 0 from the query comes from a CC di-base, while the 0 in the reference comes from a TT di-base: it is impossible to recognize the difference.

In case of an inversion:

$$R_{gen}: \quad AAACGACTTGTTGT|aactcgtg|CACACGTCTACGCT$$

$$Q_{gen}: \quad AAACGACTTGTTG\mathbf{T}|\underline{c}acgagt\underline{t}|CACACGTCTACGCT$$

In this situation the two underlined nucleotides must be complementary in order to have the same color in the sequence or, as in the previous case, some error or SNP must change in a right way the bold nucleotide: in this case, a G must substitute the bold T[18].

On principle, it is possible to know how the nucleotides change, so what color to expect and where the right breakpoint is but one must know the exactly length of structural variation involved, its type and breakpoints. Unfortunately, it is impossible to use the data one is trying to predict: as result, color on breakpoints change semi-randomly. This adds some degree of uncertainty in the breakpoint detection when sequences in color space are used. Also the same problems that affect spliced-alignments in base space are present, so overlap, mismatches, random matches and so must taken into consideration with the color space. As a result, in breakpoint detection for color space, the error could be raised of one unit and start and end positions could be shifted of half unit:

[17] AA, CC, GG and TT are all translated into color 0

[18] di-base GC is translated into color 3, as di-bases CG, AT and TA

- in case of perfect split (no overlap between the two pieces, all nucleotides matches)
  - deletions have start and end shifted of half unit to the left, both errors raised of one unit
  - no change for insertions; color space makes more probable to have some nucleotides overlapping, so there is no difference with base space
  - inversions behave as deletion if the breakpoint covered is the second one; otherwise, start error is raised of two units, end is shifted on the left of half unit and end error is raised of one units; start doesn't change
- in case of split with overlap between the two pieces
  - deletions have start and end shifted of half unit to the left, both errors raised of one unit
  - no change for insertions; there is a small probability that color space combines with sequencing errors, SNP and/or randomness to have two pieces seeing each perfectly that detect the breakpoint with one or more base of shifting; however, this is a very rare case
  - inversions behave as deletion if the breakpoint covered is the first one; otherwise, both errors are raised by two units and start and end don't change
- in case of split with one or more nucleotide mismatching
  - no change for deletions; color space makes more probable to have some nucleotides mismatching, so there is no difference with base space
  - no change for insertions; color space makes more probable to have some nucleotides mismatching, so there is no difference with base space
  - inversions behave as deletion if the breakpoint covered is the first one; otherwise, both errors are raised by one unis and both start and end are shifted of half unit to the left

2.6 Zygosity

After the prediction of all structural variations, SV_finder estimates their zygosity. The estimation consists of counting and comparing certain type of arcs, mainly arcs that cover only a breakpoint and arcs that cover the entire structural variation. This is possible because breakpoints are detected with a base-precision, so both types of arcs are well known; otherwise it would be nearly impossible to determine without errors if an arc covers a breakpoint without covering the entire structural variation, so the estimation of how many arcs for each type are present would be wrong.

Other tools for the detection of structural variations completely forget about zygosity: they only predict start, end and type of structural variation without saying anything about the potential homozygosity of heterozygosity of it. This happens because breakpoints have a great error or because there are too few arcs:

- in order to have a good precision in breakpoint detection, the standard deviation of the library must be small, so the library itself must be short (and well made, obviously)
- estimation of zygosity is based on comparing certain type of arcs; a lot of arcs means randomness has a smaller impact on the result of the comparison
- unfortunately, short library has few arcs (low physical coverage)
- other tools need short library sequenced a lot in order to estimate zygosity with the method used by SV_finder, but this is not very cheap

SV_finder makes good use of the second step: a 'long' library[19] allows the breakpoint detection with a base precision and also give a lot of arcs to estimate zygosity. Obviously, in case of very short libraries there are few arcs and so SV_finder estimation can be full of errors.

2.6.1 Parameters

SV_finder compares certain types of arcs and so it needs to know how the comparison must be done. Two parameters is needed:

[19] a long library is a library that has at least about $50x$ arc coverage

- a number $ArcTol$ to set tolerance in comparing arcs
- a number $ArcMin$ to set minimum number of arcs to consider a position i as covered

The first number is used for deletions and inversions; the estimation happens with a comparison of two number of arcs and a tolerance is needed in order to avoid problems with randomness. For example, when a deletion is in heterozygosis the two type of arcs are ideally in equal numbers, but in practice there is always a certain degree of randomness that make the type different numerous.

The second number is used for insertions; the estimation happens by verifying if an insertion breakpoint is covered, thus proving that there are at least one chromosome without insertion. SV_finder must know how many arcs are needed to consider a positions as covered in order to do this.

2.6.2 Deletion

A deletion is a piece of DNA that lacks from the query genome but is present on the reference genome; for this reason, an arc that cover the only breakpoint on the query genome aligns at great distance on the reference genome. On the contrary, if no deletion is present all arcs behave normally, so they align with a distance around the average length of the library. Based on this observations, from the chromosome affected by the deletion come out arcs with a great distance, while from the wild chromosome come out right arcs (figure 2.19). SV_finder must recognize if an arc A is a great arc arc_D or a right arcs arc_B in order to assign zygosity to a deletion del:

- del has start S_{del} and end E_{del} with errors of Er_S e Er_E respectively
- A cover positions from i to j, so its distance d is $d = j - i + 1$
- A is uniquely aligned with right orientation

Under the above conditions

- if
  - $d > AvgLen + 3 * StdDev$
  - $i \leq S_{del} - Er_S$

  - $E_{del} + Er_E \leq j$

  A is count as an arc_D

- if

  - $AvgLen - 3 * StdDev \leq d \leq AvgLen + 3 * StdDev$
  - $i \leq S_{del} - Er_S$
  - $S_{del} + Er_S \leq j \leq E_{del} - Er_E$

  or

  - $AvgLen - 3 * StdDev \leq d \leq AvgLen + 3 * StdDev$
  - $S_{del} + Er_S \leq i \leq E_{del} - Er_E$
  - $E_{del} + Er_E \leq j$

  A is count as an arc_B

- because arc_B comes from two point of the genome (the two breakpoints of the deletion on the reference genome) and arc_D comes from only one point (the only breakpoint of the deletion on the query genome), arc_B is finally divided by 2 to normalization reasons.

If all is correctly done, arc_D is the number of arcs coming from the affected chromosome and arc_B is the number of arcs coming from the wild chromosome. Comparing the two numbers decides zygosity of *del*: in case of deletion in homozygosity, both the chromosomes are affected by the deletion, so ideally there aren't arcs arc_B and all arcs are arc_D; in the opposite case, one chromosome is affected so it 'produces' arc arc_D and one is wild so it 'produces' arc arc_B:

- if $arc_B - ArcTol \leq arc_D \leq arc_B + ArcTol$, deletion is predicted as heterozygosis
- if $arc_B + ArcTol < arc_D$, deletion is predicted as homozygosis
- if $arc_D < arc_B - ArcTol$, deletion zygosis is unknown

The third case can seem very strange: mainly arcs from wild chromosome are found and this suggests that only wild chromosomes are present; but, if no affected chromosome is present, the predicted deletion is a false positive. This is a possible explanation, but there are other reasons: randomness, problems during alignments, low arc coverage, small deletion are all possible explanations. Moreover, in the simulations run there was absolute no connection between unknown zygosity and false positives, so the output of SV_finder is simply an unknown zygosity without further modifications.

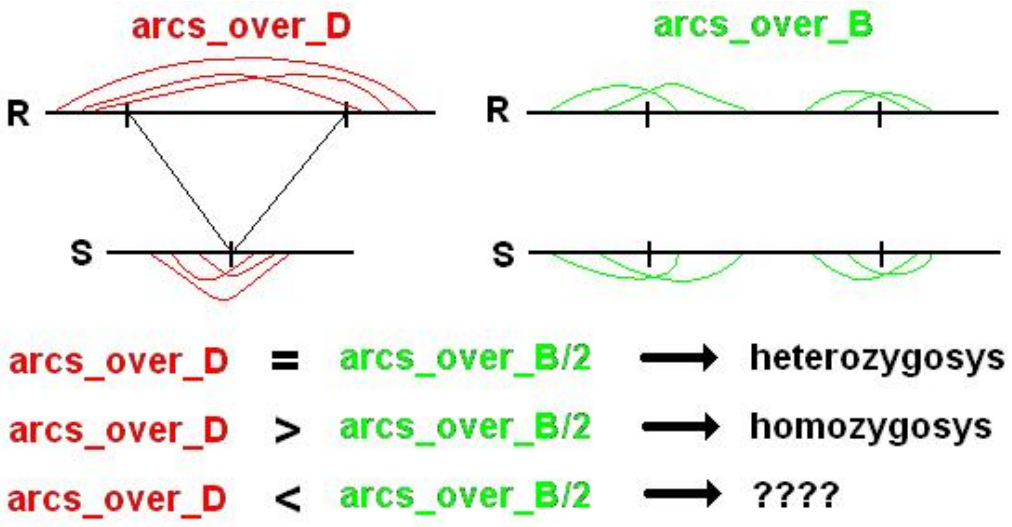

Figure 2.19: The wild chromosome produces arcs that cover only on breakpoint, while the affected chromosome produces arcs that cover the entire deletion: counting and comparing the amount of these two type of arc is the way to compute zygosity.

2.6.3 Insertion

An insertion is a piece of DNA present present only in the query genome: it could a novel sequence of DNA or a sequence present in a total different position on the reference genome. When an insertion occurs, there is only a breakpoint on the query genome and this point can or cannot be covered by arcs, depending on library length and insertion length: if the library is

too short of the insertion, there aren't arcs long enough to entirely cover the insertion. Based on this observation, from the chromosome affected by the insertion no (or very close) arcs come out, while from the wild chromosome come out right arcs (figure 2.20). SV_finder simply takes all arcs A that come from the wild chromosome and decide if their number arc_I is big enough to assume that a wild chromosome is present in order to assign zygosity to an insertion ins:

- ins has start S_{ins} with error of Er_S
- A cover positions from i to j, so its distance d is $d = j - i + 1$
- A is uniquely aligned with right orientation

Under the above conditions

- if
  - $AvgLen - StdDev \leq d \leq AvgLen + StdDev$
  - $i \leq S_{ins} - Er_S$
  - $S_{ins} + Er_S \leq j$

  A is count as an arc_I

If all is correctly done, arc_I is the number of arcs coming from the wild chromosome. If arc_I is big enough SV_finder assumes that a wild chromosome is present:

- if $ArcMin < arc_I$, insertion is predicted as heterozygosis
- if $ArcMin \leq arc_I$, insertion is predicted as homozygosis

For insertions the considered arcs are those only one standard deviation within the library average length, not 3 standard deviation. This is because if the insertion is so short (respect the library) that some arcs cover its breakpoint on the query genome even on the affected chromosome, there is an overestimation of arc_I that make the program predict too much heterozygosis insertions. To avoid this problem, the requirements for right arcs are made stricter.

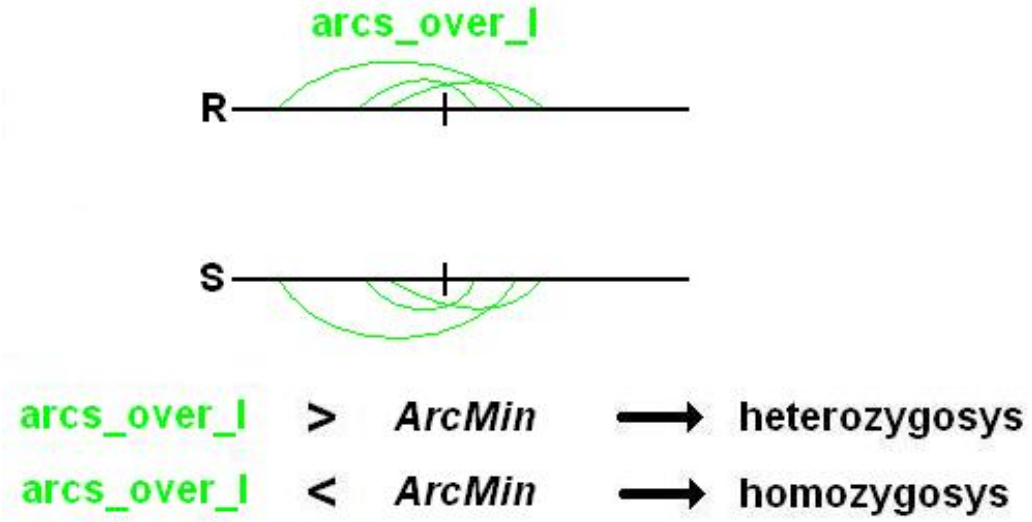

Figure 2.20: The wild chromosome produces arcs that cover the insertion point on the reference, while the affected chromosome cannot: if this amount is greater than a given parameter, *ArcMin*, insertion is hetetozygous, otherwise it is homozygous.

2.6.4 Inversion

An inversion is a piece of DNA that is reversed in the query genome: on the reference genome there are strand + and strand −; on the query genome, on strand + we found strand − of reversed DNA piece, and viceversa. If an inversion is present, an arc that cover only one of its breakpoints[20] align with wrong strand at whatever distance; if no inversion is present, only right arc are present. Based on this observations, from the chromosome affected by the inversion come out arcs with a wrong orientation, while from the wild chromosome come out right arcs (figure 2.21). SV_finder must recognize if an arc A is a wrong orientation arc arc_W or a right arcs arc_R in order to assign zygosity to an inversion inv:

- inv has start S_{inv} and end E_{inv} with errors of Er_S e Er_E respectively
- A cover positions from i to j, so its distance d is $d = j - i + 1$

[20]reads that anchor the arc align one inside the inversion and one outside

- A is uniquely aligned
- i and j must be
  - $i \leq S_{del} - Er_S$
  - $S_{del} + Er_S \leq j \leq E_{del} - Er_E$

  or
  - $S_{del} + Er_S \leq i \leq E_{del} - Er_E$
  - $E_{del} + Er_E \leq j$

Under the above conditions

- if strand of A is wrong[21], A is count as an arc_W
- if
  - strand of A is right
  - $AvgLen - 3 * StdDev \leq d \leq AvgLen + 3 * StdDev$

  A is count as an arc_R
- because $ArcTol$ is used and, for deletions, the number of arc_B was normalized for one breakpoint, arc_W and arc_R are both divided by 2 and thus normalized as arc_B

If all is correctly done, arc_W is the number of arcs coming from the affected chromosome and arc_R is the number of arcs coming from the wild chromosome. Comparing the two numbers decides zygosity of inv: in case of inversion in homozygosity, both the chromosomes are affected by the inversion, so ideally there aren't arcs arc_R and all arcs are arc_W; in the opposite case, one chromosome is affected so it 'produces' arc arc_W and one is wild so it 'produces' arc arc_R:

- if $arc_R - ArcTol \leq arc_W \leq arc_R + ArcTol$, inversion is predicted as heterozygosis
- if $arc_R + ArcTol < arc_W$, inversion is predicted as homozygosis

[21] it is important to remember that strand is wrong or right depending from the library

- if $arc_W < arc_R - ArcTol$, inversion zygosis is unknown

For the third case, the same considerations for deletions (last lines of section 2.6.2) are true.

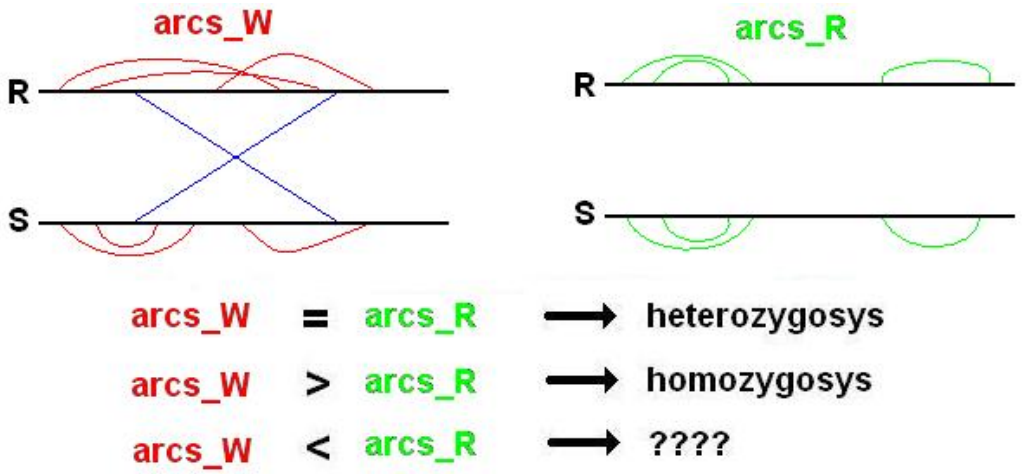

Figure 2.21: The wild chromosome (on the right) produces arcs with right strand, while the chromosome affected by the inversion (on the left, inversion is marked with red lines) produces arcs with wrong strand: counting and comparing the amount of these two type of arc is the way to compute zygosity.

2.7 Output Data

During the running, SV_finder creates many files containing values for every position of the reference genome, potential structural variations found during the first step, distribution of the values (useful to repeat the analysis with threshold values manually computed), a lot of parameters about the library and its alignments and, obviously, a file that has all the structural variations found. This file is what really interests the user, but it has so many data inside that it can be disorientating: in human genome, thousands of structural variations are found, and the only information about them is where they locate, what type of rearrangements they are, what reads identify them. But, what are the most probably structural variations, the ones to focus in the beginning? Which genes do they interest? Where should I start from? These

are the first questions that rise up. As previously stated, the file with the list of all structural variations found is only the first step of a subsequent analysis; preparing the library, sequencing it and finding all the structural variations is a long process that is only needed to create the input data for a certain project. In section 4.2 I suggest some ideas to help studying all the data produced.

The file with the structural variations is a simple tab format, with 7 to 10 fields; each line is a structural variation:

<type> <chromo> <start_position> <start_error> <start_reads> <end_position[22]> <end_error[23]> <end_reads[24]> <zygosity> <spliced-alignments positions>

The meaning of all field:

type : type of the structural variations; it could be 'deletion', 'insertion' or 'inversion'

chromo : chromosome on which structural si found

start_position : position from which the structural variation starts

start_error : error of start position

start_reads : number of reads that confirm the start found

end_position : position to which the structural variation arrives; insertions do not have a second breakpoint, so no end_position

end_error : error of end position; insertions do not have a second breakpoint, so no end_error

end_reads : number of reads that confirm the end found; insertions do not have a second breakpoint, so no end_reads

zygosity : zygosity of the structural variations found; it could be 'homozygosis', 'heterozygosis' or 'unknow'

[22]only for deletions and inversions
[23]only for deletions and inversions
[24]only for deletions and inversions

spliced-alignments positions : sometimes, due to repeats, spliced-alignments detect more positions for the breakpoints, and SV_finder merges all the positions and give them a big error; in order to have a base precision, every potential position and the read detecting it are saved, so later it is possible to find the right breakpoint with a base precision

2.8 Useful Tools

During the development of SV_finder I needed some kind of data in order to correct errors, test ideas, write C++ code, so I developed some programs. These programs work together to create an input data for SV_finder with some known structural variations inside, and to verify how well the program detects them. The entire work-flow is a simulation, with certain features and parameters, and I also used a lot of simulations (see section 3.1) for the final testing of SV_finder.

simula_reads.cpp

The most important program I developed to help me is *simula_reads.cpp*[25]; it takes a genome, introduces in it some structural variations and finally simulates a sequenced mate-pair library[26]. It can be enhanced in a lot of ways:

- used genome can be of any length, with any number of chromosome in fasta or multifasta format
- used genome can be a real genome or a simulated one
- it is possible to introduce SNPs into the genome in order to make the simulation truer
- structural variations introduced can be of certain length, zygosity and type, all defined by the user

[25] the executable file is *simula_reads.exe*

[26] it's a mate-pair only because two sibling reads are 'sequenced' in the same direction; however, because there are simulated data, it has no sense to speak about paired-end or mate-pair libraries

- it is possible to introduce structural variation in a random way, or give in input a file with a list of precise rearrangements to introduce (can be useful to simulate real structural variations affecting human race)
- output reads has a fixed length, decided by the user
- for reads of 50bp, sequencing error are simulated, according to the space of reads: for base space, Illumina error rate is used, while SOLiD error rate is used in case of reads in color space
- it is possible to simulate data in base or color space
- sequence coverage is chosen by user
- distance distribution of the mate-pair library is chosen by the user

All the above features are decided by the user in order to make a simulation that fits to his purpose. It is important to note that, whatever not specified, all is random: so, for example, positions of structural variations are randomly chosen, their length and zygosity are random, positions of SNPs are random. The program prints its output on some files: beyond the files containing forward and reverse reads, there are other files with:

- a list of all structural variations introduced, their position, their sequence, their zygosity
- where and what SNPs are introduced
- where every read comes from (start and end position in the unmodified genome)
- a simple description of how must align all reads that cover a breakpoints; this is very useful when the user want to verify they ability of SV_finder to correctly spliced-align the reads and translate splice-alignments into breakpoint detected

SV_count.cpp

In order to verify how SV_finder detects structural variations, I need to compare the list of rearrangements introduced with the list of what is found. Both lists could be very long and it is very inefficient to do the comparison

manually, so I create a program, *SV_count.cpp*[27], to do the comparison automatically. This program takes in input file with the list of all structural variation introduced, file with the potential structural variations found after the first step and file with the final structural variations found after the second step of SV_finder and count true positive, false positive, false negative and how many times the zygosity was correctly predicted for every type of structural variation. Results of the comparison are stored in different files: again, one file contains a summary of all result and is the file most wanted by a user, while the other files have more specific information and could be ignored.

sim.pl

Many programs are needed to run a simulation and they have many parameters. To make my life easier, I wrote a simple perl program that takes some parameters, computes obvious ones and runs all the necessary programs. By launching *sim.pl* with the right parameters, a simulation takes place with the wanted features.

Other Tools

I also used some programs already existing to run simulations: SV_finder needs reads aligned and classified to work so, during a simulation, after read creation two other programs do that: *pass* aligns the reads and *pass_pair* takes the output alignments and classified the sibling reads using the distance, the strand over which a read aligns and the number of hits. Those two programs were developed in the lab where I did my PhD[4].

[27] the executable file is *SV_count.exe*

Chapter 3

Results and Discussion

3.1 Simulations

The first results I obtained during the PhD come from a set of simulations run in order to have a general assessment about the performances of SV_-finder. Simulations were made using programs written in C++ and Perl Code, under a Linux environment; they required a simple personal computer of average level.

3.1.1 Why?

In order to develop SV_finder I had made a lot of simulations, with tools both already existing and developed for that specific purpose. Simulations have several advantages:

replace real data : finding a structural variation with physical techniques like a PCR or a western blot is very difficult. In the past, some were used (section 1.6) but all of them had problems, the most important were the inability to detect short structural variations, certain types of them, or in numbers high enough to use some statistical approaches; this limits very strongly the real data available to test a tool like SV_-finder, because basically there isn't yet a genome with all structural variations affecting it well detected. Two solutions are made to solve the lacking of real data: one is running simulations, the other is to take a sequenced genome, predict structural variations with the tool tested and finally compare results with predictions of another tool, possibly

a well known tool which works well. But in the latter solution, one compares two predictions, so if the tools interested use very different algorithms, then the comparison could be good, but if they use the same idea, the fact that they predict the same things does not mean that they both work well: they can predict the same things and make the same mistakes.

simulate real world : a simulation is, as the name suggests, a way to make the outside world run into a virtual environment. It is well known that there are limits on the way we can simulate the world, both physical (Heisenberg uncertainty principle) and practical (a lot of more powerful computers are needed, as well as mathematical ways to solve Schrodinger equation when it is more complex than the case of a hydrogen atom), but in case of structural variations a simulation works well. In a simulations there are some constraints, for example on the length and zygosity of structural variations introduced, but the rest is randomly chosen: where and how change the genome and, most important, where to put the sequenced reads. It is assumed that sequences cover a genome using a Poisson distribution, but in a genome there are regions impossible to sequence, repeats, regions where is very difficult to align something or, on the contrary, a lot of sequences align due to the low complexity of the region itself. All this aspects of a genome cannot be well reproduced in a simulations: using a real genome helps, but it is unknown that the sequencing machines could have some sort of bias that make them sequence only certain region, or sequence different regions in different ways, or else. Everyone assumes that a sequencing machine chooses randomly the region to sequence, so if one sequence many regions many times[1], then he obtain a uniform coverage over the entire genome used.

known structural variations : maybe this is the most important advantage; when a simulation is done, the user decides how to modify the genome that will be virtually 'sequenced' and it is known where a modification takes place. Also it is possible to create a lot of situations: how does a tool behave with only deletions? How does it behave with only structural variations of 100bp? What are the best parameters to obtain certain structural variations? Because the place and type of all

[1] as happens with the NGS machine: billions of sequence are made!

structural variations is known, it is possible to compare the results with what was introduced to see what is wrong, see where the tool can be improved, try new ideas and, finally, test the performances of the developed tool.

A simulation gives an controlled environment to work and, as all virtual environments, sometimes is not very similar to the real world, so there are also some drawbacks:

randomness : something that is simulated as random is **not** random. Genome sequence should be random, but we all know that there some big unbalance, for example, the fact that the GC content is meaningful, but also the nucleotide order in the sequence has a very depth meaning, not yet full understood. Sequences are thought to randomly cover the genome and being produced by the sequencing machine, but it is know that it is not completely true. The problem is that it is unknown the 'randomness level': are only very few sequences not random? Are almost all sequence not random? Or half of them? Without knowing the answer, we can only assume that all sequences are random (as they appear random) and work under this assumption.

structural variations introduced : it's very hard to simulate a real situations, because it is still unknown how a real situation is. Again, one can assume that a genome is interested by all structural variation types, roughly in the same quantity and with a great range of length. Maybe deletions are more frequent than inversions, due the the higher probability that a piece of DNA is cut away and that's all, while an inversion required the same cutting and then a rejoin of the piece in a reversed way. Maybe length of insertion is not random: Alu and other trasposons contribute a lot to create insertions of a certain length. One thing is sure: there are structural variations of any type and length, so for example a genome cannot be affected only by deletions of 100bp. It can be useful to run simulations with very specific characteristics, but these simulations don't reflect the real situation. In my simulations I always introduced all the structural variation types used with a great range of length in order to recreate a situation the most possible suitable to mime the real world.

after all, it's always a simulation : no matter how you simulate the real situation, no matter how good are the algorithms used, no matter how

you do things, your simulation is only a simulation, ever. It can give a great view on how the program works in general, it can show you how a program could work on real data and it can even show you how to improve results on that real data, but a simulation cannot be the final proof. The program must be prove on a real data, to see what it's the output, if it makes senses, if it is what to expect. Also it is a good idea to compare your tool with others of the same type, maybe on the simulations: in real data, you cannot say if a common prediction is right or rather a common error, but in a simulation this is possible.

Due to the above advantages and drawback, I used a certain strategies to show that SV_finder works very well:

1. first I used simulations to see how SV_finder works
2. then, on the same simulations, I compare it with another tool for structural variation detection (see sections 3.2 and 1.6)
3. lastly I run SV_finder on a real sequenced genome to see what was detected and if that makes sense

3.1.2 How Simulations Were Made

A simulation is a workflow made up of different programs that work together, each one using the output of the previous as input. The script perl *sim.pl* executes the workflow: it takes the input parameters, gives them to the different programs and takes their output. The work-flow is the following steps:

1. *simula_reads.exe* creates the simulated mate-pair library
2. *pass* aligns forward e reverse reads
3. *pass_pair* classifies the sibling reads
4. *SV_finder.exe* detects deletions, insertions and inversions
5. *SV_count.exe* compares structural variations introduced with what is found

Parameters of SV_finder are default ones; pass_pair used general parameters that depends from the library (which is always known) and pass did alignments not specific. The most important parameters are the ones for simula_reads: they decide how much the simulation is like what happens in real situations:

- the reference genome used is the human chromosome 2 of UCSC hg19 assembly; all the lines containing at least one 'N'[2] are cut out to avoid underestimation of sensibility; chromosome 2 was used because, among all the human chromosomes, is the one with the highest number of detected bases
- in every simulation 500 deletions, 500 insertions and 500 inversion were introduced with the following features:
  - random position with no overlapping structural variation
  - random length between 1 and 10000bp
  - random zygosity (half rearrangements were homozygosis, half were heterozygosis)
- about 238000 SNPs, roughly one every 1000bp, in random positions, to simulate differences between reads and reference not due to sequencing errors
- error sequencing, using SOLiD rate for simulation in color space and Illumima rate for simulation in base space
- simulated reads of 50bp: at the moment, this length is the shorter possible with NGS machines
- mate-pair libraries follow a real library (figure 3.1)
  - 'pesco' library in green line, with an average length of 593 and a standard deviation of 63
  - 'casonato' library in red line, with an average length of 1612 and a standard deviation of 375
  - 'pomodoro' library in blue line, with an average length of 8184 and a standard deviation of 1094

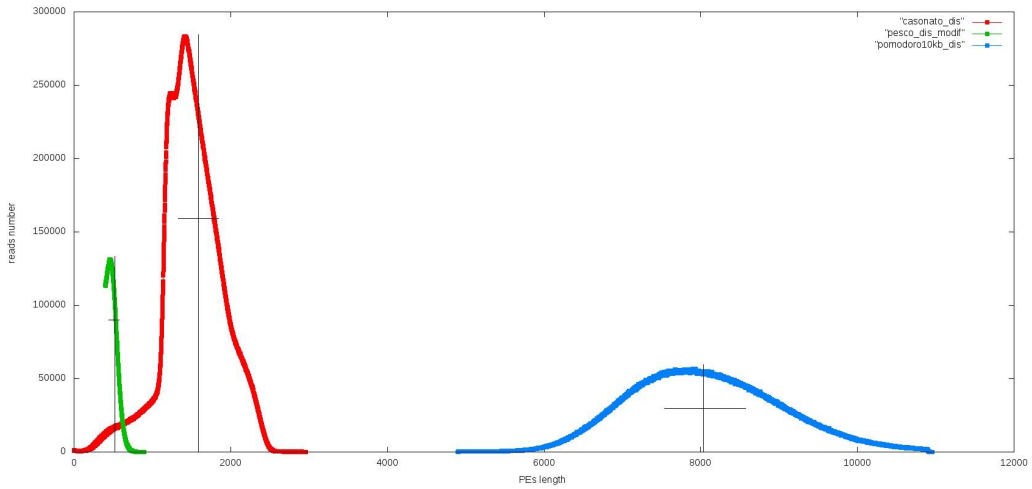

Figure 3.1: Real libraries used simulations; crosses show average length (vertical lines) and standard deviation (horizontal lines); these real distributions were chosen in order to evaluate SV_finder with short, medium and long distributions.

Every simulation uses the same previous parameters, but different library, sequence coverage and sequence space[3]:

different library : the libraries showed in figure 3.1 are examples of short, medium and long distribution; other tools want low standard deviation to well locate breakpoints, so they need short libraries; with long and even medium libraries, they cannot works well. However, SV_finder relies of the spliced-alignments to find breakpoints with a base precision, so it can be used even with those libraries.

different coverage : coverage is very important; in general, having more sequences means improving results. For SV_finder, a higher sequence coverage is a bigger number of arcs[4], so better results in the first step. But more arcs means better results also for other tool. However, higher sequence coverage improves dramatically the second step,

[2]a nucleotide not determined

[3]base or color space

[4]which is related also from the physical coverage: more reads and/or longer libraries mean both a higher physical and arc coverage

with its spliced-alignment, so SV_finder gains more benefits than other tools as the sequence coverage increases.

different sequence space : SV_finder detects structural variations from sequences both in base space and color space; this means that it can works with SOLiD, Illumina and 454 data, although the latter is not very suitable for structural variations discovery due to the lower output.

In total simulations try 3 libraries, 6 sequence coverage (1x, 2x, 4x, 8x, 16x, 32x) and 2 sequence space, for a total of 36 simulations ($3*6*2 = 36$). When a simulation was finished, SV_count computed

- sensibility after the first step
- percentage of false positive after the first step
- sensibility after the second step
- percentage of false positive after the second step
- fraction of SV with right predicted zygosity after the second step[5]

A structural variation is considered as correctly found only when its real breakpoints are within the prediction positions and errors; otherwise, the structural variations is considered as not found.

3.1.3 Graphics

Simulations with same distribution and sequence space but different coverage are grouped together; results are then showed for each type of structural variation, for a total of 18 graphics. Title graphic has 3 words:

- first word indicates the distribution used in simulation and could be 'pesco', 'casonato' or 'pomodoro'
- second word refers to the type of structural variation considered and could be deletion, insertion or inversion
- third word refers to the sequence space and could be base or color space

[5] during the first step non zygosity estimation takes place

On the x axis the sequence coverage is showed; these coverages start at 1x up to 32x, doubling the value each time. On the y axis the percentage of each result is showed:

- sensibility is scaled from 0-1 to 0-100
- percentage of false positives is not scaled
- percentage of right zygosity is not scaled

Every graphic has 5 lines:

green : sensibility after the first step; it shows how well SV_finder detects regions affected by a certain type of structural variation

purple : percentage of false positive after the first step; it shows the fraction of predictions made after the first step that are false positive; a high value can be a lot of false positives but also a small number of overall predictions

blue : sensibility after the second step; same as green line, but only predictions after the second step are considered

orange : percentage of false positive after the second step; same as purple line, but only predictions after the second step are considered

yellow : percentage of structural variations which zygosity is correctly predicted (after the second step); it shows the fraction of structural variations with right predicted zygosity among all the structural variations that are true positives (false positives are not considered)

All the 18 graphics are shower in pictures 3.2, 3.3 and 3.4; in section C, results are showed as tables.

3.1.4 Comments

In general, simulations show that SV_finder works very well; at $16x$ sequence coverage, all structural variation types are correctly predicted with a sensibility greater than 90% and a very low number of false positives. With greater sequence coverages, sensibility increases of few points. At very low

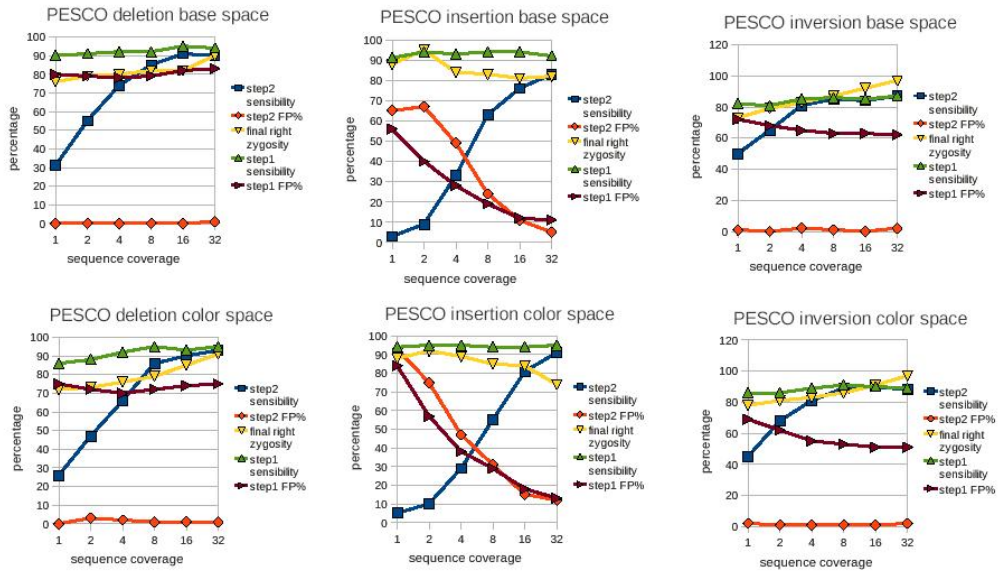

Figure 3.2: Simulation results in case of 'pesco' distribution, so with a short library

levels of sequence coverage, even $1x$, the program works and achieves 5% to 40% of sensibility, depending from the type of structural variation.

As previously stated, SV_finder uses two steps to detect structural variations. The first one is based on physical coverage and is quite similar to methods of the state of the art. Physical coverage is usually high ($> 100x$), because libraries used are long and/or a lot of reads are produced, but in both cases there are some problems. In the first one, a long library has also a great standard deviation for its distribution, thus resulting in a poor localization of breakpoints and in difficulties in detecting too short structural variations. In case of a lot of reads produced with a short library (which means low standard deviation), the costs of sequencing increases. Existing methods requires highly sequenced libraries with low standard deviations, so they could perform well in highly-costing data and could not work in low-coverage data. Moreover, methods of the state of the art suffers from false positive, because they are all base upon the fact that an arc is wrong[6] only in the presence of a structural variation. This is not completely true: there

[6]with 'wrong' meaning that its length and/or strand are not what is expected

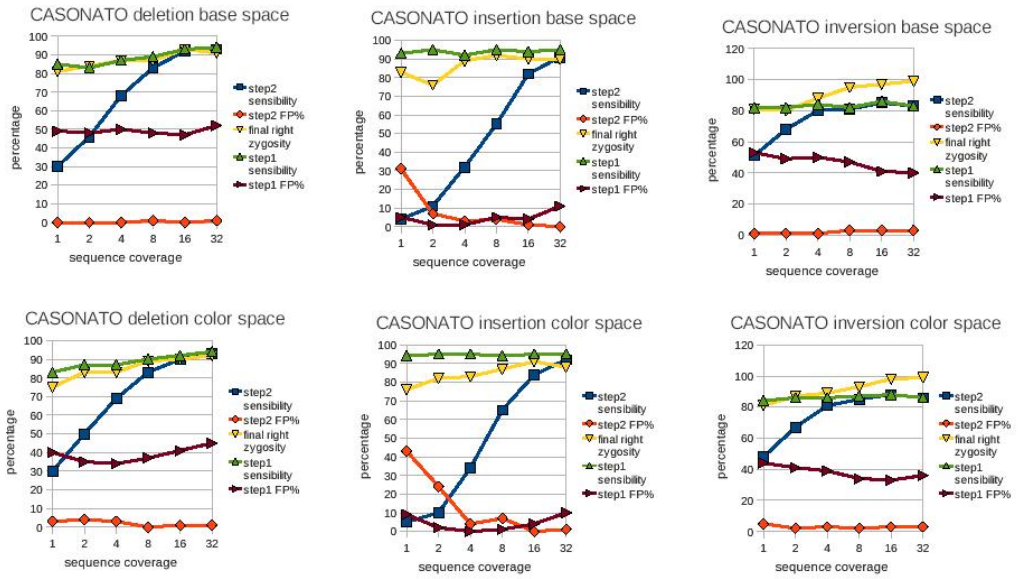

Figure 3.3: Simulation results in case of 'casonato' distribution, so with a medium library

are other reasons for the arc wrongness, for example biases in the process of creating the library, sequencing errors, problems during alignment process. All of these reasons are still not very known and scientists detect structural variations under some assumptions, for example fragmentation of DNA during creation of the library is completely random and there aren't preferential points of breaking in the DNA. Also there are a lot of alignment tools and they differ for sensibility, sensitivity, speed, behavior on repeats. Most tools for detecting structural variations don't consider all these aspects so, where they found a minimum number of arcs differing from expected ones, they claim the structural variation; these methods differ in the meaning they give to the word 'differing'.

The second step of SV_finder was developed in order to solve the problem of false positive and, meantime, find out the correct breakpoints with base-precision: a spliced-alignment of certain reads is done in order to detect breakpoints even with great standard deviation and back up the structural variation itself with two independent evidences, one from 'wrong' arcs and one from the splice-alignment. By doing this, almost all false positives found

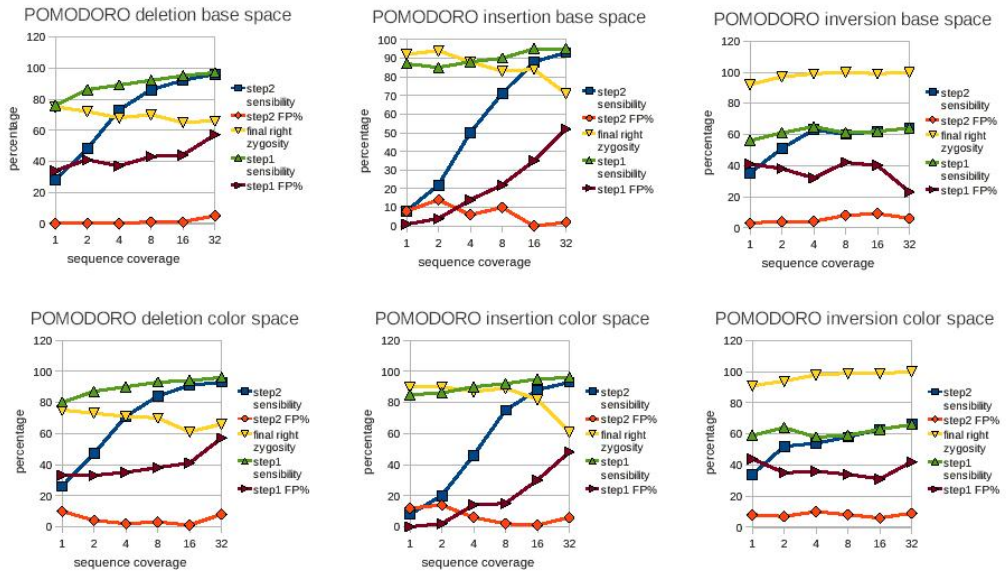

Figure 3.4: Simulation results in case of 'pomodoro' distribution, so with a long library

in the first step are cut down, as shown by the orange line in graphics 3.2, 3.3 and 3.4. It is important to note that the percentage of false positive is computed as number of wrong structural variations divided by number of all predicted structural variations so, if few of theme are finally predicted, it is possibile that the percentage of false positives is high. This happens for insertions: in the simulations with a $1x$ sequence coverage, very few insertions are predicted, about 50; if 15 of them are false positives, then a percentage of 30% for false positives results. But if the number of predicted insertions is higher, for example 300, then the percentage of false is 5%, that is more reasonable. In results for insertion at $1x$ sequence coverage of 'pesco' library (figure 3.2) and 'casonato' library (figure 3.3), the percentage of false positive is about $70-100\%$ and $30-40\%$ respectively, but the final sensibility is very low, around 10%: this means that the real number of false positive is low, around 30, among 500 real insertions simulated.

The second step also allows to lower the requirements for the first step; this great helps the detection of structural variations too short or supported by too few data; Many other tools use strict parameters in order to avoid

too many false positives: they only consider arcs that seem truly different from what to expect and skip over dubious arcs. This avoid false positives, but also avoid those structural variations that, due to a short length or randomness, produce only arcs not entirely wrong, so on most cases other tools cannot detect all structural variations affecting the sequenced genome. The second step of SV_finder is designed cut down a false positive, so in the first step it can consider also the situation in which only dubious arcs are present: if dubious arcs arise from a false positive, the second step will find out, otherwise the structural variation that produce those arcs will be found.

The second step is based on sequence coverage, on the probability that a read covers breakpoints (see section 2.5.1). At $1x$ sequence coverage, if there is a read R of length L, and if a splice-alignment with a minimum anchor of A[7] is done, in order to have R covering a breakpoint, the breakpoint must lies between positions A and LA of R. In simulations, $L = 50$, $A = 6$, so a read must cover a breakpoint between positions 6 and 44. Even at $1x$ sequence coverage, Poisson distribution and simulations show that this happens in a good percentage of cases, about one times every three; moreover the probability of such an event increases as sequence coverage increases, and our simulations show that at $3x$ sequence coverage a lot of reads cover breakpoints and correctly detect structural variations, achieving 50% of sensibility. It is a very good level of sensibility for a mere $3x$ sequence coverage, if one thinks about that other tools require much more higher coverages, up to $40x$ (for example VariationHunter, section 1.6), to work.

In simulations 'pomodoro' library (simulations in figure 3.4) the final sensibility for inversions (blue line of both graphics in the last column) never goes over a 65%: even with a very high sequence coverage, $32x$, about one third of the inversions are not found. The problem lies on the length of 'pomodoro' library: as discussed in section 2.4.3, an inversion signal is directly proportional to its length; long inversions give a great signal[8], while short inversions give low signal[9]. Because of length of 'pomodoro' library, the default threshold value for inversions is very high (see section 2.4.5), so the combination of low signal and high threshold value is a lowering of final sensibility. The same happens, at a minor rate, for simulations using 'casonato' library (figure 3.3), where the final sensibility cannot go over 80%: in this case, due

[7] where 'minimum anchor' is the minimum length of a read to be considered as aligned

[8] a 'great signal' means that there are a lot of arcs that align with wrong strand

[9] a 'low signal' means that there are few arcs that align with wrong strand

to 'casonato' library being shorter than 'pomodoro' one, the threshold value is smaller, so lower signal can be detected. For simulations using 'pesco' library (figure 3.2), this problem seems almost disappeared: the sensibility reached for inversions is about 85%, a value very close to the sensibility of 90% achieved for deletions and insertions. It is important to note that the problem arises only when the default value for inversions is used, so a used who wants better sensibility can lower the threshold value and achieve high sensibility for inversions.

The detection with base-precision of breakpoints makes SV_finder capable to compute zygosity of each structural variation. Calculation of zygosity is based upon counting certain type of arcs and comparing their numbers. These types of arcs are basically arcs that cover only one breakpoint and arcs that cover the entire structural variation but, if its breakpoints are not well defined, the count of these arcs is not enough accurate, so the results are not well accurate. This is the main reason for which other tools don't predict zygosity. Thanks to the second step, breakpointa are accurately detected and it is possible to well count the different type of arcs: simulations show that, in about 80% of right structural variations, zygosity is correctly predicted.

Finally, simulation results show no difference between base and color space: results are basically the same in sensibility. In general, these is a slightly higher number of false positive in color space, but this can be due to the fact that simulation of errors in color space is based upon SOLiD data, while simulation of errors in base space is based upon Illumina data, and SOLiD has a higher error rate than Illumina. Another difference is the precision of breakpoint detection: as discussed in section 2.5.6, the color space adds a certain degree of uncertainty in where a true breakpoint lies, so the error given to starts and ends of all types of structural variation are bigger in case of color space.

3.2 Comparison with VariationHunter

In the same simulations used to show how SV_finder works, another tool for structural variation detection was run: VariationHunter (section 1.6.1). In tables 3.2 and 3.2 comparison results are shown.

Deletions				
Conditions	TP VH	FP VH	TP SV	FP SV
I pesco 1	6	351460	158	1
I pesco 2	11	445269	277	1
I casonato 2	40	209193	234	1
I casonato 4	56	245649	342	1
I pomodoro 1	130	42614	141	1
I pomodoro 2	153	45731	242	1
I pomodoro 32	135	42456	482	28
S pesco 4	19	541908	333	8
S pesco 16	6	388534	452	7
S casonato 1	25	175754	153	6
S casonato 16	33	177694	454	9
S casonato 32	0	232	466	9
S pomodoro 1	157	42790	134	16
S pomodoro 2	153	44039	239	12
S pomodoro 32	60	30222	469	43

Insertions				
Conditions	TP VH	FP VH	TP SV	FP SV
I pesco 1	0	0	17	32
I pesco 2	0	0	48	101
I casonato 2	0	7	59	5
I casonato 4	0	18	164	6
I pomodoro 1	0	0	42	4
I pomodoro 2	0	0	114	20
I pomodoro 32	0	1	468	16
S pesco 4	0	0	145	129
S pesco 16	0	0	407	74
S casonato 1	0	1	25	19
S casonato 16	0	0	423	4
S casonato 32	0	0	463	9
S pomodoro 1	0	0	44	6
S pomodoro 2	0	1	100	17
S pomodoro 32	0	0	469	32

Each table show number of true positives (TP VH) and number of false

positives (FP VH) found by VariationHunter and number of true positives (TP SV) and number of false positives (FP SV) found by SV_finder. The column 'Conditions' shows the simulation type: for example, a 'I pesco 1' means than simulation was in base space, with 'pesco' library used and a $1x$ sequence coverage. By looking at the comparison, three things are very clear:

- some simulations are lacking: for evaluation of SV_finder, a total of 36 simulations was made, but here only 14 are present
- most of the simulations show a 0 for insertion results of VariationHunter
- no results is shown for inversions

A lot of simulations lack because it was not possible to make VariationHunter do a prediction for them. On the run, it gave errors or it froze somewhere during the execution of its algorithm: in both cases, the prediction was not obtainable. Honestly, I don't know the reason why VariationHunter did not work in all simulations: data was always the same[10] and hardware had not problems; also, many tries was made, so it is very unlikely that every time there was a hardware problem. The only possible explanation is a problem with the software: maybe VariationHunter is not well designed or is unable to work with different data.

No inversion was predicted by VariationHunter: in its manual it is stated that VariationHunter only works when sibling reads with wrong strand are on the same strand[11]; simulations were made with sibling reads that had wrong strand when both reads were on different strand, so this could be the reason. The only problem is that in the input file for VariationHunter no strand is present: a list of mate-pairs is given as input, and it is declared only if strand is right or wrong, so VariationHunter don't know anything about real strand. Again, it could be an error of mine in using VariationHunter or (more probably) VariationHunter is not able to detect inversions with the used library.

3.2.1 Problems with VariationHunter

VariationHunter is not a user-friendly tool for some reasons:

[10] same format, I mean
[11] they are '++' or '- -'

it is not parametric : its last version was made to work only on a (unknown) version of human genome, so a modification on the program was needed to make work it on the human chromosome 2.

it does not use complete path for files : VariationHunter assumes that some needed files stay in the same directory of the main script; moreover, all files that it creates are in the directory from which the program is used, so 2 different process with VariationHunter must be launched from different directory in order to avoid conflict between created files.

it has a lot of steps to compute the files for the analysis : 9 different scripts constitute VariationHunter, most of which simply take a file, read it, and re-print its content on a different file with something changed or added; instead of reading input file only one time and modify its content in one run, modifications are made in multiple steps, thus spending a lot of time and increasing the risk of an error. In fact, in many simulations VariationHunter blocked in one of these steps: after dozens of hours it even hadn't read a small file with some thousands of lines.

3.2.2 VariationHunter vs SV_finder

Beside the lacking of a lot of simulations, the present ones give a good view of how another tool works in comparison to SV_finder, because the simulations shown cover all the situations: base and color space, short, average and long libraries and different sequence coverages (only the $8x$ lacks). The comparison gives reasonable answers, only all possible combinations lack. Two things are very clear:

high number of false positives : VariationHunter predicts in each simulation a very high number of false positives, thousand and thousand of deletions found that do not exist; it find them because there is one or more arcs with a wrong distance, but they do not arise from a structural variations. They arise from something else: a sequencing bias, a not random DNA fragmentation, alignment reasons. This suggests that tool for the structural variation detection find too many false positives, because every tool is based on the same idea of VariationHunter: a wrong arcs mean a structural variation.

low number of true positives : VariationHunter finds a fraction of all the structural variation present; in case of deletions, 500 of them were introduced in the query genome. In the best case, about one third of the deletions were correctly found, but with over forty thousand of false positives: it is easy to guess something if one does many and many tries. VariationHunter was used with non strict parameters, so one could expect a lot of false positives (but not so many); however, one must expect a loto of true positives, but this did not happen.

Another things (not shown) is the breakpoint resolution: whereas SV_finder found breakpoint with a base precision, VariationHunter had error of order of magnitude of a thousand bases. It's a very big error: the reason why is that VariationHunter, as all tools unlike SV_finder, needs library with a low standard deviation to find breakpoint with a great resolution, otherwise they are poorly found.

Comparison show that SV_finder works very in data that are very challenging for other tools. Moreover, with low sequence coverage[12], SV_finder obtains good results, while other tools had awful predictions.

3.3 Results of Real Data

SV_finder was used on real data: the genome of an individual affected by a blood coagulation disease was sequenced with SOLiD and SV_finder was used on the output. This disease was not lethal, nor it has severe consequences on the general health of the carrier; also it was known to be due to a genetic variation. The problem was that the affected gene was impossible to find, so a bioinformatic approach was tried. The main idea was to sequence the genome and find all the differences with the reference human genome, considering that the affected gene should be found among these differences.

From the genome of this individual 2 different libraries was done, but they had a so similar average length and standard deviation that they were used as if they were a single library. SV_finder tolerates small differences within the library because the approach used is based on splice-alignment, whereas other programs need a perfect and very narrow insert-size distribution. However, the libraries were made with different version of SOLiD: first

[12]SV_finder can work well with a $10x$ sequence coverage, while VariationHunter was testes on a $42x$ sequence coverage

one consisted of reads 25bp length, while the second consisted of reads 50bp length. First library was sequenced 3 times, second 2 times, for a total of $11x$ sequence coverage: the minimum required coverage to obtain good sensibility, according to simulations (section 3.1). However, sequences of 25bp do not perform very well in splice-alignments and they constitute about a $3x$ sequence coverage: more or less, the effective sequence was $8x$.

3.3.1 Structural Variations Found

SV_finder was used on sequenced genome of individual affected by a blood coagulation disease and it found a certain amount of structural variations showed in figure 3.5 (in section D tables with numbers are present).

In total 3366 deletions, 14574 insertions and 214 inversions were found. These seem a lot of structural variation but, as demonstrated[15][16][17] in previous studies, an adult genome is affected by thousands of structural variations, some of them even connected to mortal diseases. The exact estimation of total number is still unknown, due to the fact that structural variations experimentally characterized are very few and, instead, they are detected with *in silico* techniques, but:

- a lot of types of structural variation to be detected
- a lot of different tools for structural variation detection
- tool cannot be tested on real data, because a sequence genome with all structural variations experimentally characterized doesn't exist yet, so the performances of every tool are not totally assured

Different studies report different amount of structural variation in a human genome, from a thousand to dozens of thousands. The number found by SV_finder fits well with was it is expected; currently, this is the only information about the good quality of the prediction. In section 4.2 a suggestion is made to use and prove all the data.

It is interesting to notice that the number of insertions found is very high, while inversions found is a very small fraction of all structural variations found. The low amount of inversions found is expected: inversion is an event less probable and more difficult to find than a deletion, so finding much less

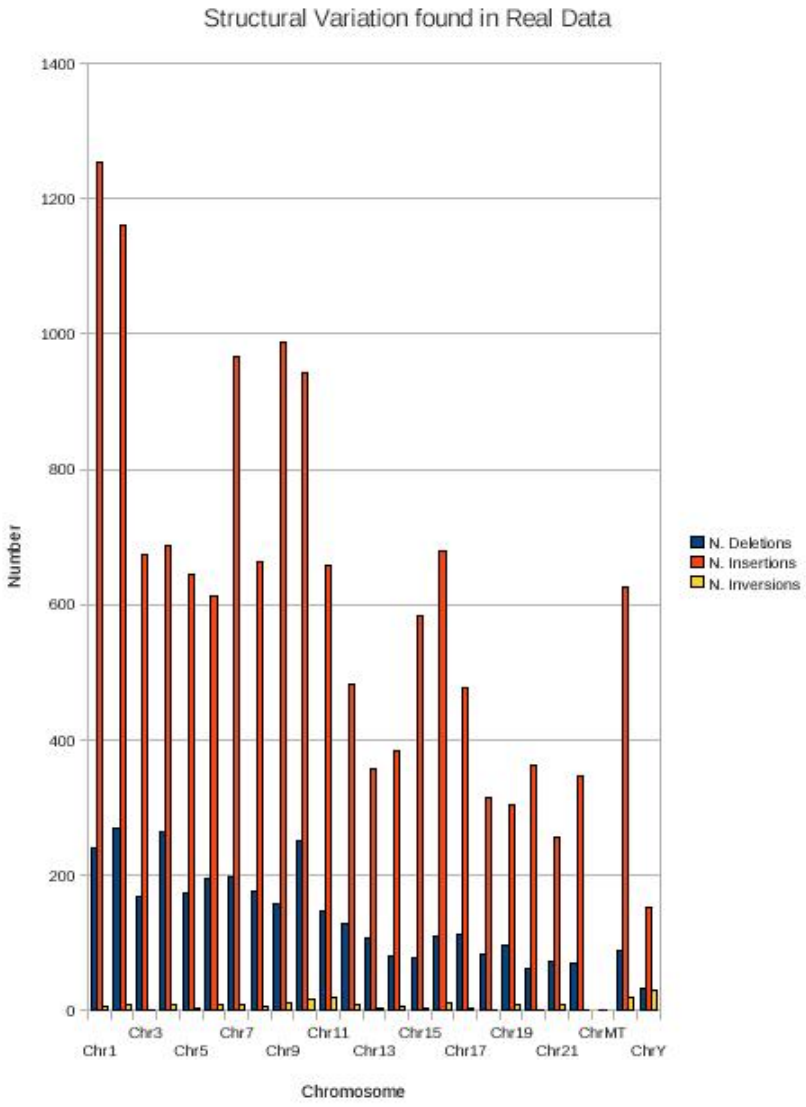

Figure 3.5: Structural variations found by SV_finder: on x axis there are the chromosomes, on the y axis the number of different variants (deletion, insertion and inversion) found in each chromosome; blue columns show number of deletions found, orange columns show number of inseretions found and yellow columns show number of inversions found.

inversions than deletions fits well. Insertions are as common as deletion[13], so the higher number of them in comparison to deletions is unexpected. As far as I know, there could be two possible explanations:

insertions are more common that deletions : beside SV_finder, no tool is able to detect insertions too long; to find an insertion they need an arc that covers it, but an arc can cover an insertion only if it is longer than the insertion itself, otherwise no arc is present for the analysis. Other tool needs very short library (in this way, standard deviation is very small and breakpoint detection is good) to work well, so they find only very short insertions. SV_finder can detect insertions of any length, regardless of library average length, because it exploit a signature peculiar to long insertions.

many insertions found are false positives : for the detection of structural variations very short reads were used; about 25% of total reads had a length of 25bp. Moreover, some reads were trimmed, so the bases in the ends with very low quality was cut away in order to improve the alignment. The trimming process make very short reads compare: the statistic reported a read minimum length of 19bp. For the splice-alignment, the read length is a critical factor; simulations show that a read length of 50bp is enough to obtain good results, but probably with shorter sequences results become worse and worse. A lot of false insertions can be due to splice-alignments of such short reads: probably, if the detection is repeated without considering too short reads, results could be improved.

3.3.2 Errors on Breakpoints

The splice-alignment approach allows to cut down the number of false positives and to detect breakpoint with a base-precision. In the simulation run, the error of breakpoints was always very small, it never became greater than a dozens of bases, but in this results some structural variations have an error of hundreds of bases (see section 4.3). How this could be possible? Many factor could be responsible for this problem, but there two main reasons:

[13]maybe they are a bit less common, but magnitude order is more or less the same: if 1000 deletions are found, one must expect to find a thousand of insertions, not only 10 or 100

genome is repeated : if a breakpoint is on a sequence repeated in a near spot and the two location fall into the same potential structural variations (found in the first step), then during the splice-alignment both location are considered as breakpoint. SV_finder computes a unique location for this breakpoint usign as value the average of the location and as error an half of the distance between them. So, if two locations are at 1000bp of distance, then the breakpoint will have an error of 500bp. However, SV_finder reports the locations from which the final position for a breakpoint is computed, so in the refinement step of result (section 4.2) it is possible to evaluate the best location, thus lowering a lot the error.

reads are short : a long read splice-aligns more difficult than a short one, so in case of short reads they splice-align easily, thus giving birth to wrong alignment, so a breakpoint can have multiple locations due to splice-alignments of short reads; multiple locations means great error on breakpoint detection. As reported in previous section 3.3.1, a lot of short reads are present, so probably repeating the prediction without them will dramatically reduce breakpoint errors in most cases.

Chapter 4

Conclusion

4.1 General Performance

The aim of my PhD research was to investigate the possibility to develop a bioinformatic tool for the detection of structural variations, using paired-ends and mate-pairs sequencing data. As a result I created SV_finder and I tested it under different conditions. My conclusion is that SV_finder seems to works very well:

- simulations show that sensibility is high with very few false positives
- comparison with VariationHunter shows that at very low sequence coverages SV_finder performs better than any other state of the art tools
- results of real data show that SV_finder can be efficiently used in real projects to detect structural variations affecting the sequenced genome

SV_finder is not limited by platform technology nor by the library used: it works in base and color space, with short, medium and long libraries, with paired-ends and mate-pairs. Breakpoints are always found with base precision and their identification does not depend by the standard deviation of the library sequenced. At $16x$ sequence coverage, the program achieves high sensibility, with very few false positives. As now, a $16x$ sequence coverage is easily achieved by almost all NGS machines available, so SV_finder can work at its best.

The sensibility does not reach 100% due to a main reason: the presence of very short structural variations in simulations. As already stated (section 3.1.2), in every simulation there are structural variations with a range

of lengths from 1 to 10000bp, so a certain number of very short structural variation is present. Short rearrangements are still a challenge, because they produce few arcs that most of the times produce a signal lower than background noise. Moreover, it seems that there is a range of lengths, roughly between 10 and 50bp, in which a structural variation cannot be detected: it is too long to use alignments of a fragment library[1], but it is too short to change in a noticeable way the arc distribution. So, small structural variations remain unsolved for every available tool. It is important to note that other methods have a minimum length for the detection higher than the theoretical one for SV_finder, and also they cannot detect insertions longer than the longest available arc[2] so, although SV_finder has some limitations, it performs better than other methods.

Read length is a critical factor: longer reads mean better results. In simulations, a standard read length of 50bp was used, because at the moment of writing this PhD thesis it is the minimum length for NGS machines[3]. If read length is too short, for example as happened with the first versions of SOLiD sequencer, which produced read of 25bp, the chance of a read covering a breakpoint decreases, so more reads are needed; however, shorter reads mean weaker alignments, so more false positives and greater errors on breakpoint positions. All this means that the second step probably does not work so well with very short reads, so SV_finder has to rely of the first step to detect structural variations. In this case it works as well as other tools of the state of the art.

An important part is also played by the rightness of reads: errors in reads covering breakpoints can lead to errors on the detection of breakpoint itself, so in case of high sequencing error rate, breakpoints are detected with bigger errors; however, errors increased of only few base, so there is still a base-precision detection.

[1] if reads are very long, about hundreds of bases, it is possible to use the splice-alignment against all the reference to detect structural variations

[2] which depends from average length and standard deviation of the sequence library

[3] well, in some cases a paired-end or mate-pair library is made of sibling reads of 75bp and 35bp, but it is only slightly different to have equally long sibling reads of 50bp

4.2 Final Results: What to Do with Them

In general, a tool for structural variation detection gives in output a list of rearrangements. The first question that arises in user's mind is: and now? What can I do with this information? How can I use all the output data? The problem is not trivial, because there are so much information to simply lose own aim, own goal. The first thing that one can think is to manually analyse all data, but there is too much data for a manual analysis, from hundreds to thousands of regions. This is a great number: it is very unlikely that one single person (or, even, a team of people) takes one region, studies it and then improves its detection. It is too much work to be done. The obvious solution is to automate the process of human refinement, but if it can be automated, why don't include it in the tool itself? The answer is that the refinement process cannot be completely automated; it has to be made by a human being, but some sort of automation is possible to speed up the entire process.

First of all, one raw solution is to help the human user to see at once all the informations about a certain region needed for the refinement; this informations could be:

- start and end of structural variations: these are the most important information to know for obvious reasons
- sequence and arc coverage: always aligned reads and arcs must agree with the structural variation found; an homozygous deletion cannot have too much reads aligned inside
- repeat index: a number that gives an idea on how much repeats there are in studied region is always useful, because repeat regions can be responsible for great errors in breakpoint, false positive and the lack of expected reads and/or arcs
- structural variations known to affect the studied region: a deletion found where there is an already known deletion is always welcome

The above are general informations useful in nearly all the situations; the idea behind is simple: for example, I have a deletion; it can arise from a certain reason, different from really having a structural variation, so I visualize all what is known about that reason and, based on what I see, I decide if the deletion is a false positive or not, if start and end are right, if the errors

on breakpoints can be lowered. Obviously, the user must be skilled in order to really improve the detection. In figure 4.1 is showed an example on how visualize and refine structural variation first output.

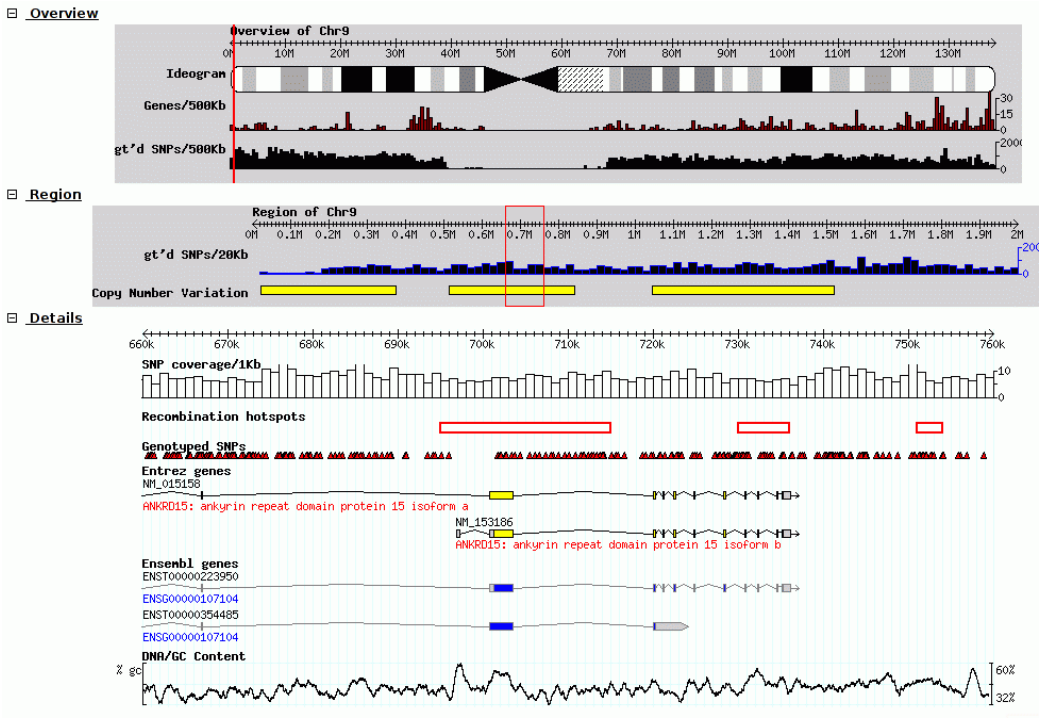

Figure 4.1: An example of how visualize structural variations for refinement using GBrowser (http://gmod.org/wiki/GBrowse), a very powerful tool in bioinformatic.

This solution is very useful, but still time-consuming: although speed refinement is greatly improved for each structural variations, there are thousands of them, so the overall time is still high. What can be done now? In the majority of cases, the user is interested in structural variations with certain features, that can be:

- type of structural variation: for example, the user wants only inversions
- precise regions that can be affected by a rearrangement
- certain genes that can be affected by a rearrangement

- certain portion of genes that can be affected by a rearrangement
- certain chromosomes

With the above cases, user focuses his attention only on few structural variations, those ones that affect the regions or genes[4] interested, so the number of rearrangements to study become manageable. This can happen mostly because the user known something *at priori* about what he is interested. In general the best solution is having a system that select only certain structural variations and focusing attention on them, skipping all the rest.

4.3 Problems

Structural variation detection suffers from some problems that currently are not yet solved. Repeat regions are the most difficult problem: since the beginning of the sequencing era, repeats needed extra work to be solved because reads covering a repeat cluster together; to solve them, a sequence longer then the repeat region is needed, but this can be difficult in case of long repeat regions.

In structural variation detection, repeats make some sibling reads align in different positions in the reference, thus creating a lot of combinations and arcs. These arcs differs each others in the distance: here lies the problem. Two sibling reads connect two positions and the length of the connection depends mainly from the presence of a structural variation; if one or both sibling reads has more than one positions (which means more than one alignment), there are more connections, more arcs; some on them can have a right length, some cannot: how can one choose which is 'right' and which is 'wrong'? A wrong arc can arise from the presence of a structural variation or a repeat, but there is no way to know this, so repeat regions remains almost an unexplored field in terms of structural variations.

The above is a common problem for all tools for structural variation detection, SV_finder suffers from repeat in another way. The second step uses a spliced-alignments to detect breakpoints with a base precision; if small[5] repeats are present, one or both of the read pieces align in more that one position, so multiple positions are found for a single breakpoint (see section

[4]that can be a single gene or a certain gene region, such as UTR or coding region or other

[5]repeat of dozens of base, so with a length of the same magnitude order of read length

3.3). SV_finder clusters all positions into a single one with a great error, and this is the breakpoint: the problem is that its error has magnitude order of hundreds of nucleotide, so the base precision promised is not reached. This can be solved in refinement process (section 4.2): all the potential positions is known so the real breakpoint is not spread across hundreds of places but rather between few positions; with other informations, such as sequence and arc coverages, it is possible to choose the right position with a great reliability.

Sometimes breakpoint are not well defined; for example, an inversion happens when:

1. DNA breaks in two different positions that are approximately near (not further that hundred thousands bases)
2. the segment between the breaks reverses
3. the segment is reinserted in the same place but in the reverse direction

During the second stage an partial degradation of DNA ends can happen, so when the segment is reinserted, some nucleotides around the two breakpoints are missed. A read covering one breakpoint align against the reference in a wrong way:

reference genome

R_{gen} : $AAACGACTTGTTGT|tagcgatccgat|CACACGTCTACGCT$

R_{read} : $TTGT|\quad cgat|$

$R_{degread}$: $TGTT\quad|\quad tccg\quad|$

query genome with partial degradation

Q_{gen} : $AAACGACTTGTT\underline{GT}|\underline{at}cggatcgcta|CACACGTCTACGCT$

$Q_{degread}$: $TGTT\quad|\quad cgga\quad|$

query genome

Q_{gen} : $AAACGACTTGTTGT|atcggatcgcta|CACACGTCTACGCT$

Q_{read} : $TTGT|atcg$ $|$

The underline nucleotides in the second schema are degrated so, instead of getting a $TTGTatcg$ as read covering the breakpoint, a $TGTTcgga$ read is gotten. The 4 lacking bases make inversion start and end not defined[6]. Sometimes the degradation could be so severe that a splice-alignment is no longer possible, so the structural variation could not be detected.

In the first step, SV_finder creates 5 indexes and, using threshold values, translate them into a list of potential structural variations. On concept, this is similar to the tools of the state of the art, but is less complex, simpler. The reason behind is the presence of the second step, so in the first one it is only needed to detect the region affected by a structural variation. Anyway, this can lead to some problems:

- unnecessary false positives: a false positive can arise because an arc is wrong for other reasons and not for the presence of a structural variation; but for randomness very small regions ('small' means 'less than 100bp length') can show a lot of wrong arcs, so some regions can be wrongly recognized as potential structural variations
- structural variations skipped: structural variations which signal is lower than threshold value are missed (as described in sections 2.4.3 and 3.1.4)
- length of potential structural variations depends from the threshold values: lowering a threshol value has two consequences:
  - more potential structural variations are found
  - already found potential structural variations are longer (see figure 4.2)

An improvement to the first step could lead to an overall improvement of SV_finder (section 4.4).

[6]it is important to note that this is a complex structural variation: inversion flanked by two very small deletions, so it can be a difficult case to solve; most other tools cannot solve it

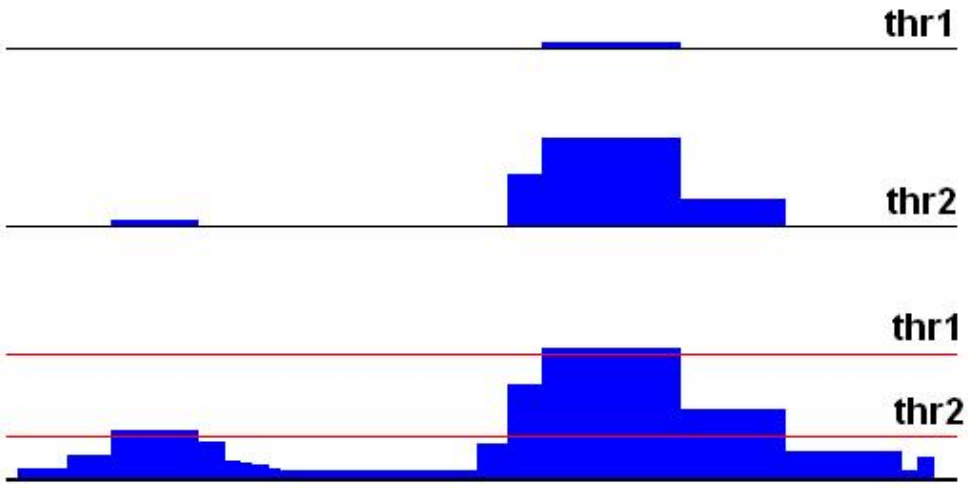

Figure 4.2: Lowering the threshold value leds to more and larger potential structural variation in the first step.

SV_finder estimates the zygosity of each structural variation found; it is something that most other tools do not do, but it is not yet a perfect computation: as showed by simulations (section 3.1), zygosity is correctly predicted about four times out of five. Different structural variation types behave different and also at different sequence coverage results are different:

deletions : according to simulations, percentage of right zygosity never reaches 100%; for 'pesco' and 'casonato' libraries, it increases with the increasing of sequence coverage, while for 'pomodoro' library it slowly decreases! Also it seems that the longer the library the worse the result is, so for a good computation short libraries with a lot of arcs are needed. As described in section 2.6.2, SV_finder takes arcs covering the entire deletion and arcs covering only one of the two breakpoints in the reference genome, because the firsts mainly come from the affected chromosome (i.e. the one with the deletion), while the seconds mainly come from the wild chromosome (the one without the deletion). The problem is that the dividing line between these two types are less and less defined as the deletion length decreases: in case of

short deletions([7]), from the wild chromosome both arcs covering the entire deletion and arcs covering only one breakpoint could come out, so the arc count can be wrong. The reason why with 'pomodoro' library percentage of right zygosity decreases with the increasing of sequence coverage is unknown: maybe it is due to the high standard deviation of the library, so the counting of arcs has always errors and some degree of randomness; or it can be due to wrong standard parameters used for such a long library.

insertions : as for deletions, percentage of right zygosity never reaches 100% and for 'pesco' and 'pomodoro' libraries it decreases with the increasing of sequence coverage, while for 'casonato' library it increases with irregular progress. Insertion zygosity is the most difficult to compute: as described in section 2.6.3, SV_finder takes arcs covering the entire insertion (i.e. the only breakpoint in the reference genome) and compare them to a minimum number: if arcs are more than this number, it means that a wild chromosome is present, so insertion is heterozygosis; otherwise, both chromosome are affected by the insertion, so it is homozygosis. The problem arises with the minimum number and the arcs to be considered: minimum number used in simulation was default one, so it is possible that a better value could be used to improve zygosity detection; and arcs covering insertion point are very difficult to take, because a lot of background noise is present. Infact only arcs within one standard deviation to the average length are considered, not within three standard deviations, that is a more correct value (see section 1.5).

inversions : all inversions are found with correctly zygosity at $32x$ sequence coverage, because there is no uncertainty about arcs; an arc has two distinct possibility: it comes from the wild chromosome (i.e. chromosome without the inversion), so it aligns with right strand, or it comes from the chromosome affected by the inversion, so it aligns with wrong strand. Using the strand, an arc can be assigned to each type absolutely without any doubt. For inversion zygosity, amount is important: more arcs means more precision for zygosity prediction. This is the reason for a 100% of correctly found zygosity is reached and why with

[7] it is important to remember that 'short' here means 'short' compared to library length: the same deletion could be short for 'pomodoro' library, but long for 'pesco' library, for example

long library ('pomodoro' over 'casonato' over 'pesco' libraries) the zygosity prediction is better: long library means more physical coverage, so more arcs for the computation.

4.4 Possible Improvements

SV_finder will benefit for longer reads and more arcs:

- longer reads leads to an improvement of the second step, because longer reads means more nucleotides that match in the spliced-alignment, so it is stronger
- more arcs leads to an improvement of the first step, because having more arcs means less false positives and shorter structural variation can be detected

In any case, more arcs is an improvement for all tool for structural variation detection, while longer reads is beneficial only for SV_detect.

A major improvement can be done on the the first step. As simulations showed (section 3.1), the sensibility of the second step is a curve that try to reach the sensibility of the first step. The reason why is obvious: in the first step, regions affected that a structural variations are found; if that region is really affected, splice-alignment shows that and also find the right breakpoints. But if a structural variation is skipped in the first step, no spliced-alignment is done. First step is a double-edge sword: it allows to do splice-alignment in a very specific way, but on the other hand there is always the risk to lose something.

Final sensibility cannot go above sensibility after the first step, so improving the first step means improving the overall sensibility of SV_finder without adding too many false positives.

Appendix A

Acronyms

bp base pairs

DNA DeoxyNucleic Acid

GFF General Feature Format

NGS Next Generation Sequencing

RNA RiboNucleic Acid

SAM Sequence Alignment/Map

SNP Single Nucleotide Polymorphis

SV Structural Variations

Appendix B

Definition

arc : in a paired-end or mate-pair library, two sibling reads align at certain places in the reference genome; if both reads have unique alignments, there is only one arc, otherwise, in case of multiple alignment, all the combinations must be considered and each one is an arc; an alignment of a reads and an alignment of its sibling form an arc, ideally connecting two far positions on the reference genome and imposing a constraint over them, either of length or strand or both

distribution : here a frequency distribution of how many times a certain distance is found for two sibling reads aligned; in this situation, 'library' and 'distribution' refer to the same thing and are used as synonyms

hit : an alignment of a read is a 'hit'; thus, a read that align only one times has one hit, while a read that align in many places has multiple hits

library : a pool of sequenced fragments produced with a certain technique that make the name (mate-pair libraries, fragments libraries, etc.); when the sequence fragments of a paired-end or mate-pair library are aligned against a reference, they form a distribution of distance, where the distance is the number of nucleotides between the alignments of two sibling reads; in this situation, 'library' and 'distribution' refer to the same thing and are used as synonyms

read : a single sequence produced by a NGS machine is commonly called 'read'

reference genome : the genome against which one or more sequences are aligned

query genome : the genome from which come the sequences that, typically, are aligned against a reference genome

run : the output of a NGS as SOLiD is commonly referred to as a 'run', so a run is basically a pool di sequences

sibling reads : in a paired-end or mate-pair library, only the ends of each DNA fragment are sequenced; these ends are two sibling reads that must 'place' themselves at a certain distance, corresponding to the length of the original fragment; often, one read is called 'forward' read, and its sibling is called 'reverse' read

Appendix C

Simulation Tables

Field meaning:

seq cov : sequence coverage

sens s2 : sensibility after the second step

% f pos s2 : percentage of false positives after the second step

% right zyg : percentage of structural variations with right zygosity after the second step

sens s1 : sensibility after the first step

% f pos s1 : percentage of false positives after the first step

Pesco, Deletions, Base Space					
seq cov	sens s2	% f pos s2	% right zyg	sens s1	% f pos s1
1x	31	0	76	90	80
2x	55	0	79	91	79
4x	74	0	80	92	78
8x	85	0	82	92	79
16x	91	0	82	95	82
32x	90	1	90	94	83

Pesco, Insertions, Base Space					
seq cov	sens s2	% f pos s2	% right zyg	sens s1	% f pos s1
1x	3	65	88	91	56
2x	9	67	95	94	40
4x	33	49	84	93	28
8x	63	24	83	94	19
16x	76	11	81	94	12
32x	83	5	82	92	11

Pesco, Inversions, Base Space					
seq cov	sens s2	% f pos s2	% right zyg	sens s1	% f pos s1
1x	50	1	73	82	72
2x	65	0	79	81	68
4x	81	2	83	85	65
8x	85	1	87	86	63
16x	84	0	92	85	63
32x	87	2	97	87	62

Pesco, Deletions, Color Space					
seq cov	sens s2	% f pos s2	% right zyg	sens s1	% f pos s1
1x	26	0	72	86	75
2x	47	3	73	88	72
4x	66	2	76	92	70
8x	86	1	79	95	72
16x	90	1	85	93	74
32x	93	1	91	95	75

Pesco, Insertions, Color Space					
seq cov	sens s2	% f pos s2	% right zyg	sens s1	% f pos s1
1x	5	93	88	94	84
2x	10	75	92	95	57
4x	29	47	89	95	38
8x	55	31	85	94	29
16x	81	15	84	94	18
32x	91	12	74	95	13

Pesco, Inversions, Color Space					
seq cov	sens s2	% f pos s2	% right zyg	sens s1	% f pos s1
1x	45	2	78	86	69
2x	68	1	81	86	62
4x	81	1	83	89	55
8x	89	1	86	91	53
16x	90	1	91	90	51
32x	88	2	97	89	51

Casonato, Deletions, Base Space					
seq cov	sens s2	% f pos s2	% right zyg	sens s1	% f pos s1
1x	30	0	81	85	49
2x	46	0	84	83	48
4x	68	0	87	87	50
8x	83	1	87	89	48
16x	92	0	93	93	47
32x	93	1	91	94	52

Casonato, Insertions, Base Space					
seq cov	sens s2	% f pos s2	% right zyg	sens s1	% f pos s1
1x	4	31	83	93	5
2x	11	7	76	95	1
4x	32	3	89	92	1
8x	55	4	92	95	5
16x	82	1	90	94	4
32x	91	0	90	95	11

Casonato, Inversions, Base Space					
seq cov	sens s2	% f pos s2	% right zyg	sens s1	% f pos s1
1x	51	1	81	82	53
2x	68	1	80	82	49
4x	80	1	88	84	50
8x	81	3	95	82	47
16x	85	3	97	86	41
32x	83	3	99	83	40

Casonato, Deletions, Color Space					
seq cov	sens s2	% f pos s2	% right zyg	sens s1	% f pos s1
1x	30	3	75	83	40
2x	50	4	83	87	35
4x	69	3	83	87	34
8x	83	0	89	90	37
16x	90	1	90	92	41
32x	93	1	92	94	45

Casonato, Insertions, Color Space					
seq cov	sens s2	% f pos s2	% right zyg	sens s1	% f pos s1
1x	5	43	76	94	9
2x	10	24	82	95	2
4x	34	4	83	95	0
8x	65	7	87	94	1
16x	84	0	91	95	4
32x	92	1	88	95	10

Casonato, Inversions, Color Space					
seq cov	sens s2	% f pos s2	% right zyg	sens s1	% f pos s1
1x	48	5	81	84	44
2x	67	2	87	86	41
4x	81	3	89	86	39
8x	85	2	93	87	34
16x	88	3	98	88	33
32x	86	3	99	86	36

Pomodoro, Deletions, Base Space					
seq cov	sens s2	% f pos s2	% right zyg	sens s1	% f pos s1
1x	28	0	75	76	34
2x	48	0	72	86	41
4x	73	0	68	89	37
8x	86	1	70	92	43
16x	92	1	65	95	44
32x	96	5	66	97	57

Pomodoro, Insertions, Base Space					
seq cov	sens s2	% f pos s2	% right zyg	sens s1	% f pos s1
1x	8	8	92	87	1
2x	22	14	94	85	4
4x	50	6	88	88	14
8x	71	10	83	90	22
16x	88	0	84	95	35
32x	93	2	71	95	52

Pomodoro, Inversions, Base Space					
seq cov	sens s2	% f pos s2	% right zyg	sens s1	% f pos s1
1x	35	3	92	56	41
2x	51	4	97	61	38
4x	63	4	99	65	32
8x	60	8	100	61	42
16x	62	9	99	62	40
32x	64	6	100	64	23

Pomodoro, Deletions, Color Space					
seq cov	sens s2	% f pos s2	% right zyg	sens s1	% f pos s1
1x	26	10	75	80	33
2x	47	4	73	87	33
4x	71	2	71	90	35
8x	84	3	70	93	38
16x	91	1	61	94	41
32x	93	8	66	96	57

Pomodoro, Insertions, Color Space					
seq cov	sens s2	% f pos s2	% right zyg	sens s1	% f pos s1
1x	8	12	90	85	0
2x	20	14	90	86	2
4x	46	6	87	90	14
8x	75	2	89	92	15
16x	88	1	82	95	30
32x	93	6	61	96	48

Pomodoro, Inversions, Color Space					
seq cov	sens s2	% f pos s2	% right zyg	sens s1	% f pos s1
1x	34	8	91	59	44
2x	52	7	94	64	35
4x	54	10	98	58	36
8x	58	8	99	59	34
16x	63	6	99	63	31
32x	66	9	100	66	42

Appendix D

Real Data Tables

Number of Structural Variations Found			
Chromosome	N. Deletions	N. Insertions	N. Inversions
Chr1	240	1254	6
Chr2	270	1161	9
Chr3	169	673	2
Chr4	263	686	8
Chr5	173	645	4
Chr6	196	613	9
Chr7	198	966	10
Chr8	177	663	5
Chr9	157	989	12
Chr10	251	942	18
Chr11	147	657	20
Chr12	129	483	8
Chr13	106	356	3
Chr14	81	383	7
Chr15	77	583	3
Chr16	110	680	12
Chr17	113	478	4
Chr18	84	316	2
Chr19	98	304	9
Chr20	63	362	2

Number of Structural Variations Found			
Chromosome	N. Deletions	N. Insertions	N. Inversions
Chr21	73	255	10
Chr22	71	346	2
ChrMT	0	1	0
ChrX	88	626	19
ChrY	32	152	30

Bibliography

[1] http://www.ornl.gov/sci/techresources/Human_Genome/home.shtml

[2] C. Alkan, B, P. Coe, E. E. Eichler, *Genome structural variation discovery and genotyping*, Nature Review, 2011, Vol. 12

[3] Applied Biosystems, *Principles of Di-Base Sequencing and the Advantages of Color Space Analysis in the SOLiD^{TM} System* from site http://www.appliedbiosystems.com

[4] D. Campagna, A. Albiero, A. Bilardi, E. Caniato, C. Forcato, S. Manavski, N. Vitulo, G. Valle, *PASS: a program to align short sequences*, Bioinformatics, 2009, Vol. 25

[5] F. Hormozdiari,C. Alkan, E. E. Eichler and S. C. Sahinalp, *Combinatorial algorithms for structural variation detection in high-throughput sequenced genomes*, Genome Research, 2009, Vol. 19

[6] D. R. Bentley, S. Balasubramanian, H. P. Swerdlow, G. P. Smith, J. Milton, C. G. Brown, K. P. Hall, D. J. Evers, C. L. Barnes, H. R. Bignell, et al, *Accurate whole human genome sequencing using reversible terminator chemistry*, Nature, 2008, Vol. 456

[7] A. R Quinlan, R. A. Clark, S. Sokolova, M. L. Leibowitz, Y. Zhang, M. E. Hurles, J. C. Mell, I. M. Hall, *Genome-wide mapping and assembly of structural variant breakpoints in the mouse genome*, Genome Research, 2010, Vol. 20

[8] S. Lee, F. Hormozdiari, C. Alkan, M. Brudno, *MoDIL: detecting small indels from clone-end sequencing with mixtures of distributions*, Nature Methods, 2009, Vol. 6

[9] J. O. Korbel, A. Abyzov, X. J. Mu, N. Carriero, P. Cayting, Z. Zhang, M. Snyder, M. B. Gerstein, *PEMer: a computational framework with simulation based error models for inferring genomic structural variants from massive paired-end sequencing data*, Genome Biology, 2009, Vol. 10

[10] B. Zeitouni, V. Boeva, I. Janoueix-Lerosey, S. Loeillet, P. Legoix-né, A. Nicolas, O. Delattre, E. Barillot, *SVDetect: a tool to identify genomic structural variations from paired-end and mate-pair sequencing data*, Bioinformatics, 2010, Vol. 26

[11] F. Sanger, S. Nicklen, A. R. Coulson, *Dna sequencing with chain-terminating inhibitors*, Biotechnology, 1977, Vol 24

[12] P. Medvedev, M. Stanciu, M. Brudno, *Computational methods for discovering structural variation with next-generation sequencing*, Nature Methods Supplement, 2009, Vol. 6

[13] L. Feuk, A. R. Carson, S. W. Scherer, *Structural variation in the human genome*, Nature Review, 2006, Vol. 7

[14] L. A Weiss, Y. Shen, J. M. Korn, D. E. Arking, D. T. Miller, R. Fossdal, E. Saemundsen, H. Stefansson, M. A. Ferreira, T. Green, O. S. Platt, D. M. Ruderfer, C. A. Walsh, D. Altshuler, A. Chakravarti, R. E. Tanzi, K. Stefansson, S. L. Santangelo, J. F. Gusella, P. Sklar, B. L. Wu, M. J. Daly, *Association between Microdeletion and Microduplication at 16p11.2 and Autism*, New England Journal of Medicine, 2008, Vol. 7

[15] F. Zhang, W. Gu, M. E. Hurles, J. R. Lupski1, *Copy Number Variation in Human Health, Disease and Evolution*, Annual Review of Genomics and Human Genetic, 2009, Vol. 10

[16] , J. Wang, W. Wang, R. Li, Y. Li, G. Tian, L. Goodman, W. Fan, J. Zhang, J. Li, J. Zhang, Y. Guo, B. Feng, H. Li, Y. Lu, X. Fang, H. Liang, Z. Du, D. Li, Y. Zhao, Y. Hu, Z. Yang, H. Zheng, I. Hellmann, M. Inouye, J. Pool, X. Yi, J. Zhao, J. Duan, Y. Zhou, J. Qin, L. Ma, G. Li, Z. Yang, G. Zhang, B. Yang, C. Yu, F. Liang, W. Li, S. Li, D. Li, P. Ni, J. Ruan, Q. Li, H. Zhu, D. Liu, Z. Lu, N. Li, G. Guo, J. Zhang, J. Ye, L. Fang, Q. Hao, Q. Chen, Y. Liang, Y. Su, A. San, C. Ping, S. Yang, F. Chen, L. Li, K. Zhou, H. Zheng, Y. Ren, L. Yang, Y. Gao,

G. Yang, Z. Li, X. Feng, K. Kristiansen, G. Ka-Shu Wong, R. Nielsen, R. Durbin, L. Bolund, X. Zhang, S. Li, H. Yang, J. Wang, *The diploid genome sequence of an Asian individual*, Nature, 2009, Vol. 6

[17] S. Levy, G. Sutton, P. C. Ng, L. Feuk, A. L. Halpern, B. P. Walenz, N. Axelrod, J. Huang1, E. F. Kirkness, G. Denisov, Y. Lin, J. R. MacDonald, A. Wing Chun Pang, M. Shago, T. B. Stockwell, A. Tsiamouri, V. Bafna, V. Bansal, S. A. Kravitz, D. A. Busam, K. Y. Beeson, T. C. McIntosh, K. A. Remington, J. F. Abril, J. Gill, J. Borman, Y. Rogers, M. E. Frazier, S. W. Scherer, R. L. Strausberg, J. C. Venter, *The Diploid Genome Sequence of an Individual Human*, PLOS Biology, 2007, Vol. 5