UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

DEPARTMENT OF INFORMATION ENGINEERING

MASTER DEGREE COURSE IN
Computer Engineering
Bioinformatics

"Development and evaluation of a computational pipeline for the detection and analysis of repeat expansions from high-throughput sequencing"

Supervisor: Prof. Fabio Vandin

Graduating: Federica Vettor

ACADEMIC YEAR 2021 – 2022

Graduation date: 17/10/2022

Sommario

Le malattie da Repeat expansions (RE) sono una classe di patologie causati da espansioni nucleotidiche composte da almeno tre basi all'interno del DNA. Le RE causano disturbi genetici come la malattia di Huntington e vari tipi di Atassie. L'identificazione e l'analisi di RE avviene di norma tramite tecniche di laboratorio molecolare (i.e. PCR) anche se negli ultimi anni, l'avvento di nuove tecnologie di sequenziamento genico (NGS) ha permesso di analizzare le regioni codificanti di DNA ed anche l'intero genoma, alla ricerca di regioni polimorfiche, comprese le RE, in modo più efficiente e rapido. La patogenicità di un'espansione viene determinata confrontando il numero di ripetizioni identificate in una regione genomica con degli intervalli di normalità, specifici per la patologia in analisi. Per analizzare la grande mole di dati NGS prodotti, sono stati sviluppati diversi metodi bioinformatici con performance diverse.

L'obiettivo di questa tesi è studiare lo stato dell'arte dei software disponibili per l'identificazione di RE e successivamente sviluppare una pipeline computazionale che combini i risultati di ciascun programma, e determini la patogenicità delle RE in campioni reali, tenendo conto degli intervalli di riferimento. A tale scopo sono stati selezionati tre software di identificazione per RE. Metodologicamente, l'output dei tre tool è stato processato al fine di trovare regioni con RE in comune, che a loro volta sono state confrontate con specifici intervalli di riferimento per determinarne la patogenicità. La performance di tale metodo è stata testata su dati di esoma e di genoma sequenziati da campioni di pazienti con patologia da espansione nota.

Come risultato la pipeline ha rilevato le RE patologiche caratteristiche di ciascuna malattia, con una maggiore sensibilità nell'identificazione delle RE utilizzando l'esoma piuttosto che il genoma dello stesso campione.

L'applicazione di questo metodo all'analisi di esomi NGS da pazienti con malattie da RE, può essere di supporto in una migliore caratterizzazione di queste tipologie di malattie genetiche.

Summary

Repeat expansion (RE) diseases result from nucleotide expansions of at least three bases within the DNA. REs cause genetic disorders such as Huntington's disease and various types of Ataxias. Identification and analysis of RE are usually performed by molecular laboratory techniques (e.g. PCR) although in recent years, the advent of next generation sequencing technologies allowed the sequencing of DNA coding regions of DNA and also the entire genome, making the polymorphic regions identification, including REs, more efficient and quick. The expansion pathogenicity is determined by comparing the number of repetitions identified in a genomic region with ranges of normality, specific for the pathology. To analyse the large amount of data produced by NGS techniques, several bioinformatic methods, with different levels of performance were developed.

The aim of this thesis was to study the state of art of available softwares for REs identification and, next, to develop a computational pipeline that combines the results of each program and finds out the pathogenicity of REs in real samples, taking into account the reference ranges. For this purpose, three tools were selected. Methodologically, the nucleotide sequences were first analysed by these three tools and then their outputs were processed in order to find regions with common RE, and compared with specific reference ranges to determine their pathogenicity.

The performance of this method was evaluated both on exome and genome samples from patients with known expansion pathology. As result, the pipeline correctly detected pathological REs specific of each disease, and, of note an increased sensitivity was observed using exome data for REs identification compared to genome sequences from the same sample. The application of this method to analyse NGS exomes from patients with RE diseases may be helpful in a better characterization of this kind of genetic disorders.

Contents

Chapter 1

Introduction

A Repeat Expansion is a type of mutation in which a set of tandemly repeated sequences replicates inaccurately to increase the number of repeats. This mutation is responsible for causing any type of disorder labeled as dynamic mutations [1].

More than 40 diseases, most of which primarily affect the nervous system, are caused by expansions of repeat sequences throughout the human genome [3]. The disease pathogenesis varies depending on the repeat sequence, size and location within the genome. The number of repeats appears to predict the progression, severity, and age of onset diseases related to Repeat Expansion [3]. The most frequent target diseases are Huntington's disease (HD), different types of Ataxia Spinocerebellar (SCA) and Fragile X syndrome (FXTAS).

Huntington's disease [4] is a rare, inherited disease that causes the progressive breakdown (degeneration) of nerve cells in the brain. It usually causes movement, cognitive and psychiatric disorders with a wide spectrum of signs and symptoms. Which symptoms appear first varies greatly from person to person.

Instead, Spinocerebellar ataxia (SCA) [5] is a progressive, degenerative and genetic disease with multiple types. In general, SCAs are characterized by slowly progressive incoordination of gait and is often associated with poor coordination of hands, speech or eye movements. Also, it frequently results in atrophy of the cerebellum, loss of fine coordination of muscle movements leading to unsteady and clumsy motion, and other symptoms.

Finally, Fragile X syndrome (FXS) [6] is an inherited genetic disease that causes intellectual and developmental disabilities.

Repeat expansion is caused by slippage during DNA replication and from the formation of loop out. Short repeat expansions can occur during DNA replication and DNA repair processes like: homologous recombination, non-homologous end joining, mismatch repair or base excision repair. Each of these processes involves a DNA synthesis step in which strand slippage might occur leading with Repeat Expansion [3].

Overall, to check whether an individual is subject to a pathology due to REs, the genome sequence is first analysed to identify repeated nucleotide patterns. The identified REs are analysed according to the source region and the number of repeats. The number of repetitions should be compared with reference values to identify the unstable region of the genome [2]. These values derive from the literature although with some discrepancies between various publications. The specific ranges for the identification of diseases related to instability are divided into premutation and pathological [2].

Expanded repeats are unstable (dynamic) mutations that often change size in successive generations [3]. So, their number of repeats can grow from one generation to the next due to DNA replication slippage and unequal recombination in multiplying cells. Indeed, the premutation range of values is useful since it indicates the probability to have individuals with a RE disease in the next familiy generation. Their presence in the regulatory regions, which may alter the gene expression, is found to be associated with the respective disease phenotype. Looking at the repeat distribution, REs are found to be present in both coding and noncoding regions of the genome [7].

Therefore, in Repeat Expansion analysis, there is a certain threshold of repeats that can occur before a sequence becomes unstable. Once this threshold is reached the repeats will start to rapidly expand, causing longer expansions in future generations. There is still not enough research found to understand the molecular nature that causes thresholds, but researchers are continuing to study that the possibility could lie with the formation of the secondary structure when these repeats occur [2]. These observations have led to the hypothesis that the threshold is determined by the number of repeats that must occur to stabilize the formation of these unwanted secondary structures, due to the fact that when these structures form there is an increased number of mutations that will form in the sequence resulting in more triplet expansion [1].

Currently, several approaches were proposed to identify and analyze Repeat Expansions although their integration has never been proposed. The best solutions, found in Bioinformatics literature, are briefly reported below.

Laboratory testing

Usually detection of repeat expansions is performed with polymerase chain reaction-based assays or with Southern blots for large expansions. Also, with this method, a subsequent analysis is required since the results for certain regions are compared with the normal ranges in order to identify pathological REs among those detected.

Genetic laboratories conduct a large number of tests for Repeat Expansion disorders, but the detection rate is low, and job times are of the order of weeks or months. No comprehensive panel or testing method exists that simultaneously tests for all known repeat expansions using the current gold-standard detection methods of PCR and Southern blot.

Software approaches

The bioinformatic tools proposed to detect de novo or known repeats are:

- Alignment-free detection tools like SuperSTR (details in Section 3.1.3).
- Genome-wide detection of short tandem repeat expansions by long-read sequencing with RepeatHMM tool. RepeatHMM is an algorithm to estimate repeat counts from long-read after taking high base calling error rate into consideration. It has two main routines. RepeatHMM-scan that scans whole-genome long-read sequencing data to determine repeat counts of tens of thousands of REs. Then, we use a divide-and-conquer strategy to group REs into smaller sets, and run RepeatHMM on each set. RepeatHMM-DB that collects repeat counts for all REs in a reference genome and builds a database of normal repeat range for all available REs. Overall, RepeatHMM runs a scan module and then compares those repeat counts with the corresponding repeat ranges in RepeatHMM-DB [8].
- Alignment-like tools as ExpansionHunter [9], exSTRa [13], and TRED-PARSE [16] (details in Sections 3.1.1, 3.1.4 and 3.1.2).

Some of these methods were analysed and evaluated later, in this thesis.

The rest of this thesis is organized as follows. In Chapter 2 we describe the motivation and the objectives of this project. Then, in Chapter 3 we report some details about the tools tested and about the dataset. In Chapter 4 the pipeline is described. In Chapter 5 we show the experimental results obtained by different types of tests. Finally, We conclude the discussion in the last chapter.

Chapter 2

Objectives

Repeat expansions are common genetic variations that are normally associated with neurogenetic disorders. It is a dynamic and unstable type of mutation, and the number of repeats can vary from generation to generation.

The disorders caused by Repeat Expansion are clinically and genetically heterogeneous, and vary depending on the repeat sequence, size and location of the responsible gene.

PCR-based repeat length analysis is the most convenient method for diagnosing diseases associated with repeat expansion. However, the established laboratory techniques for diagnosing repeat expansions (PCR and Southern blot) are cumbersome, low-yield and unsuitable for parallel analysis of multiple gene regions. In fact, laboratory analysis for the identification of expansions is done specifically on the candidate gene only after the appearance of the disease phenotype. In the case of preventive analysis, on the other hand, a higher cost in terms of time and resources is required because the PCR has to be repeated on a larger group of genes related to RE diseases.

Diseases due to RE are mainly characterised by severe neurodegenerative disorders and an onset of the phenotype in adulthood. Clinical tests for the diagnosis of these particular diseases are often done too late for the application of efficient prophylaxis.

An important feature of triplet expansion diseases is the phenomenon of genetic anticipation: over generations, the disease tends to manifest itself earlier and with a more severe clinical picture. This is because the severity of the disease is linked to the number of repeats, which is unstable and tends to increase with each successive DNA replication. The higher the number of triplet repeats, the greater the likelihood of further expansion.

In view of these characteristics, there is a need for a reliable, wide-ranging method for analysing this type of mutation, taking several regions of the genome into account at the same time, with the goal to detect the presence of a possible disease quickly before its manifestation. Identifying these diseases before they occur is also important for:

- Early clinical intervention.
- Tests on parents before having children in order to check the possibility of being carriers and, thus, calculate the probability of their children being affected by specific RE diseases because the mutation increases its expansion.

In conclusion, the use of Bioinformatics systems could be an excellent method for the rapid identification and analysis of Repeat Expansions using a comprehensive view.

The main objective of this thesis is, therefore, to study the state of the art of available software for the identification of RE and subsequently develop a computational pipeline that can be used for the analysis of samples in order to identify their possible pathogenicity to diseases caused by Repeat Expansions.

Chapter 3

Tools and dataset

In this Chapter we report the tools and methods used in this project. In particular, Section 3.1 describes the tools analysed and the reasons for their use or not. In Section 3.2 we show the characteristics of the dataset used to test the tools and the pipeline developed. Finally, in Section 3.3, the pathologies of interest are briefly described with their mutational characteristics.

3.1 Tools analysis

For the identification of triplet expansion, according to some papers it is necessary to use different tools and methods to have a more complete analysis and to provide redundancy [20]. First of all, I analyse some existing softwares using a subset of the dataset (see Section 3.2). The purpose of these tests is the selection of the tools with the best performance also in relation to their theoretical approach used in the identification of REs. I report all the tools and the related reasons for their selection or not.

3.1.1 ExpansionHunter

ExpansionHunter [9, 10] is a tool for short read and it is able to detect novel repeat expansions at the locus of interest, but require, that the putative Repeat Expansions be explicitly specified in the input. This tool works well, even if the expanded repeat is larger than the read length. ExpansionHunter works on a predefined variant catalog containing genomic locations and the structure of a series of targeted loci. For each locus, the program extracts relevant reads from a binary alignment/map file and realigns them. The realigned reads are then used to genotype each variant at the locus. Genotyping is performed by analyzing the alignment paths associated with the presence or absence of each constituent allele.

Method

Its method consists of the following step.

- Whole Genome Sequencing for all of the samples analyzed.
- Identifying IRRs: to test if a read fully consists of the repeat motif we compared it to the perfect repeat sequence that was the closest match under the shifter and reverse complement operations. We defined the weighted purity score metric. We defined IRRs as reads that achieve weighted purity of 0.9 or above.
- Identifying off-target regions: IRR pairs may align to other genomic locations, especially if the RE is short in the reference genome at the target location. Identifying off target regions enables us to reduce the search for IRRs to a few regions instead of the whole genome. The mapping positions of all identified IRRs were merged if they were closer than 500 bp.
- Repeat size estimation from IRRs: We assume that the probability of observing a read starting at a given base follows the Bernoulli distribution. The starting positions of the reads occurring in a given region define a Bernoulli process and the number of reads starting in the region follows a Binomial distribution. The confidence interval for the repeat size is estimated by the parametric bootstrap method. The same procedure is used to obtain point estimates and confidence intervals for repeat sizes from flanking reads. If there is no evidence of long repeats with the same repeat unit elsewhere in the genome, both anchored in-repeat reads and in-repeat read pairs can be utilized to estimate the full length of the repeat. If multiple long repeats with the same repeat unit exist, then the size of the repeat is estimated only from anchored in repeat reads and so is capped by the fragment length. For shorter alleles, the sizes of repeats were determined using spanning reads. For repeats that are close to the read length, the repeat may be too long to produce spanning reads but too short to produce IRRs. It computes the maximum-likelihood genotype consisting of candidate repeat alleles determined by spanning, flanking, and in-repeat reads.
- Repeat size determination from spanning reads: The reads spanning the repeat are identified from all the reads that are aligned within 1kb of the target repeat region. Each of these reads is tested for the

presence of the repeat motif. To be considered spanning, a read must achieve a WP score of 0.9 across the repeat sequence and its flanks. If the flanking sequence is similar to the repeat motif then more flanking sequence is required to identify the end of the repeat and the beginning of the flanking sequence.

- Repeat genotyping: Genotype probabilities for repeats of size up to the read length are calculated using a similar model as the one used for SNPs. We use read-length sized repeats as a stand-in for repeats longer than the read length. If only one allele is expanded we estimate the full size of the repeat as described above.

Input files

The input files required by Expansion Hunter are:

- FASTA file with a reference genome.
- BAM file containing aligned reads from a PCR sample. The reads file must be sorted and indexed if using the seeking mode for the analysis.
- A variant catalog file that is a JSON array whose entries specify reference coordinates and structure of each locus that the program will analyze.

Output files

They contain information about sample parameters and analysis results summarized by locus. There are two types of output file format and for the repeats variation they report:

- VCF: chromosome, start and end position of repeats region, repeats count for each allele, information about reference locus, repeats format and motif.
- JSON: Unique variant identifier, repeat unit in the reference orientation, genotype given by the size of each repeat allele, confidence interval for each repeat allele, summary of identified spanning reads given as an array with entries (n, m) where n is the number of repeat units spanned by the flanking read and m is the number of such reads (the same for in-repeat reads and spanning reads), reference coordinates of the repeat region (chrom:start-end).

Execution

Below, in listing 3.1, there is the command to run this software on Linux OS. The execution requires a series of input parameters in addition to the files indicated above.

```
/ExpansionHunter   --reads <aligned reads BAM/CRAM file/URL>  \
--reference <reference genome FASTA file> \
--variant-catalog <JSON file specifying variants to genotype> \
--output-prefix <Prefix for the output files> \
--sex <arg>  --min-locus-coverage <int>  \
--region-extension-length <int> --analysis-mode <arg> \
--threads <int>
```

Listing 3.1: ExpansionHunter run - bash linux

After many tests with different parameters setting, I decided to choose this tool because with low time cost, good results were obtained compared to the samples chosen for the preliminary tests. Furthermore ExpansionHunter estimates the repeat size by using a parametric model but does not attempt to call repeat expansions in a probabilistic framework. ExpansionHunter was used for determining whether alleles were larger than the currently known smallest disease-causing repeat-expansion alleles. ExpansionHunter is particularly valuable for identifying disease-causing expansions because these programs leverage evidence beyond the reads that span an Repeat Expansion, enabling the genotyping of larger repeat expansions.

In Section 5.1, the choice of parameters is explained in detail.

3.1.2 STRetch

STRetch [15] uses a new reference genome with additional decoy chromosomes containing artificially long versions of all repeat motif combinations. By mapping to this modified genome, STRetch identifies reads that originated from large Repeat Expansions, containing mostly sequence, that now preferentially map to the STR decoys.

Method

Its method can be divided into the following steps:

- Reads are mapped to the reference genome with STR decoy chromosomes using BWA and SAMTOOLS. It generates a set of all possible STR repeat units in the range of 1–6 bp to produce a custom STR-aware reference genome and STR decoy reference genome is created.

- Counts the number of reads mapping to each STR decoy chromosome using bedtools. These are sorted to place together read-pairs using SAMTOOLS collate and then are extracted in fastq format using bedtools bamtofastq.
- Reads mapping to the STR decoy chromosomes are allocated to an STR locus using paired information. To determine from which STR locus the reads originated, the mates of the reads mapping to a given STR decoy chromosome are collected. The counts for each STR locus are normalized against the median coverage for that sample.
- Detecting outliers. To detect individuals with unusually large STRs, STRetch calculates an outlier score for each individual at each locus. The outlier score is a z-score or p-value. A locus is called significant if the adjusted p-value is less than 0,05.
- Median coverage over the whole genome or exome target region is calculated using goleft covmed, which is later used to normalize the counts.
- Predicts the size of the expansion using the number of reads allocated to the locus. It performs simulations of a single locus at a range of allele sizes. It fits a linear regression between the number of reads mapping to the STR decoy and the size of the allele from the simulated data.

Input files

Input files required are:

- Reference genome with STR decoy chromosomes and other metafiles like BWA indices of reference, genome file of reference, STR decoy bed file and STR positions in genome annotated bed file.
- BAM file containing aligned reads from a PCR sample.
- One of the three pipelines, depending on what type of sequencing you are doing (exome or genome) and what format your data is in (fastq or bed).

Output files

Each file have the same information like chromosome, start and end reference, repeat string, reference length (number of repeat units in the reference), locus coverage (number of STR reads assigned to that locus), outlier (z score testing for outliers), p value (it is this locus significantly expanded relative to other samples), bp Insertion (estimated size of allele in bp inserted relative to the reference) and repeat units (estimated total size of allele in repeat units).

- sample.STRs.tsv: It contains STRetch results for all STR loci with any assigned reads in that sample. If there are no reads assigned to a locus, that locus will not be reported.
- STRs.tsv: It contains STRetch results for all STR loci in any sample. If reads are only assigned for a given locus in one sample, the other samples will show 0 reads.

Execution

Below, in listing 3.2, there is the command to run this software on Linux OS. It's the example for exome sequencing and bed file.

```
STRetch/tools/bin/bpipe run \
STRetch/pipelines/STRetch_exome_bam_pipeline.groovy sample1.bam
```

Listing 3.2: STRetch run - bash linux

After many tests with different input parameters, I decide to choose this tool because of its different approach regarding ExpansionHunter even though it detects Repeat Expansions, but does genotype them reliably. Its results are reliable and I need a different method to detect REs so I can compare different kinds of results.

In Section 5.1, the choice of parameters is explained in detail.

3.1.3 SuperSTR

SuperSTR [11, 12] is an ultrafast method that does not require alignment, capable of efficiently processing DNA and RNA sequencing data. SuperSTR uses a fast, compression-based estimator of the information complexity of individual reads to select and process only those reads likely to harbor expansions. It computes the counts of repeats using a ratio computed from the ratio of the size of the read in bytes after compression to its uncompressed size. We evaluated the ratio in classifying simulated repeat-containing reads against pseudorandom sequences drawn from four background nucleotide distributions. A classifier using the ratio to identify repeat-containing reads. Reads that pass the ratio classification step are processed using the maximal approximate periodicity algorithm as implemented in mreps37 to identify repetitive substrings within each read.

Method

SuperSTR's motif screening uses a one-sided Mann-Whitney U test to evaluate whether the distribution of information scores is the same in the test group and the Diversity cohort, or whether the test group distribution is greater.

- Read Processing: Each read is compressed using zlib. C is computed as the ratio of the size in bytes of the output data to the size in bytes of the input data, Thresholds on C are supplied by the user from either the pre-provided threshold tables or user-run simulation. Reads passing this threshold are then analyzed using the Kolpakov-Kucherov algorithm. Each read identified as containing repetitive elements has their start and end location, motif, length and purity recorded.

- Post processing and outlier detection: summarize the output of the read processing methods into a list of the number of repetitive elements of each length with a specified motif for each sample. This number is normalized by library size, and per-sample and per-motif files are generated. Summarisation produces a repeat count vector and information score for each motif.

- Motif-screening and outlier analysis: using a one-sided Mann-Whitney U test as implemented in the scipy library. The null hypothesis is "randomly selected samples of interest would have larger information scores than samples from the background". We computed the p-value using permutation testing of each motif-sample pair. Motifs were reported as significant if p value was less than 0.05. Values that exceed the upper 95% confidence interval of the estimate of this percentile are reported as outliers.

The work pipeline is divided in the following steps: process reads with superSTR, build a manifest file, summarize the superSTR run, perform motif screening, perform sample-level outlier detection, visualize motif screening and outputs. Each step has a specific script to perfom it.

After many tests with different samples, I decided to not consider this tool because the method used is too different. In fact the software output are some graphs showing a statistical trend.

3.1.4 exSTRa

ExSTRa [13, 14] extracts Repeat Expansions information that stems from a particular individual and which has been identified as mapping to one of the 21 STR loci, for each read. It uses a statistical test that captures the differences between an individual who is to be tested within a cohort of affected individuals and controls. The empirical p values of the test statistics were determined via a simulation method. All p-values overall RE loci for all individuals within each cohort were assessed for approximate uniform distribution with histograms and quantile-quantile (Q-Q) plots. Raw data were visualized via empirical cumulative distribution functions (ECDFs), which display as a step function the distribution, from smallest to largest, of the amount of RE motif found in each read. For this analysis exSTRa uses R packages.

After many tests with different input parameters, I decided to not consider this tool because it has some problems at the execution time.

3.1.5 Tredparse

Tredparse [16, 17] is designed to identify each allele length at predefined RE loci by using Illumina WGS sequence data that are sampled at sufficient depth.

Method

Tredparse method is divided in the following steps:

- Haploidy inference for a given locus: It models the autosomal REs as diploid loci, allowing two alleles to be inferred per locus.
- Realignment of reads near the RE region: It realigns the reads that were mapped around the RE region extracted from the BAM file. It obtains an accurate count of the occurrences of the repeat motifs. It uses dynamic programming with the Smith-Waterman (SW) algorithm to count the number of repeats and single-instruction-multiple-data (SIMD) Smith-Waterman library for fast alignment. The SW alignment yields a series of alignments with different scores, which we then compare to determine the repeat size that corresponds to the highest score. These reads are sorted into a set of observations that are integrated in a probabilistic model for RE size inference.
- Probabilistic model for calling RE: It predicts size on the basis of evidence from spanning reads, partial reads, repeat-only reads, and spanning pairs. The spanning reads are the reads that show both left and right flanking sequences. With probability p, the read is a product of stutter noise, which is dependent on the repeat unit length K and also

the GC content of the locus. Also, the algorithm uses the Poisson distribution model for spanning reads. Partial reads do not align all the way across the repeat region and contain only one flanking sequence. The partial reads have a probability mass function of discrete uniform distribution between a single repeat unit and the true repeat length. The partial reads only show a lower bound for the number of repeat units of the underlying allele. Repeat only reads consist almost entirely of repeat units. Repeat-only reads are possible only when repeat length is the same or longer than a read length. It uses the paired-end distance for extending the prediction of allele size beyond the length of a typical sequencing read because the paired-end distance is often longer than the read length.

- Tredparse combines data using maximum-likelihood estimates and the model through a grid search.
- Confidence of RE Calls given the Inheritance Model: It computes the probability that a sample is pathological, given dominant and recessive inheritance models under the assumption of complete penetration and a point cutoff. This phase exploits likelihood function and marginal distribution.

In general, the main features of Tredparse are automatic determination of the correct ploidy level to account for X-linked and autosomal loci; re-alignment of reads, leading to a more precise counting of repeat elements; use of a full probabilistic model that incorporates four types of evidence whereas most competing software programs only consider spanning reads, and so calls are limited by read length and computation of likelihood of disease under the proper inheritance model (dominant or recessive).

Input files

Tredparse requires both the BAM file and the BAM index file to be present for fast access to each sample.

Output files

In general the output is the maximum likelihood size estimates, distributions over the number of repeats, and the associated probability of having each of the 30 RE diseases. Each file (VCF or JSON format) reports information about the location of the repeat regions (chromosome, start/end position, etc.) and information about the repeat expansion event (motif, count for each allele, types, etc.).

Execution

Below, in listing 3.3, there is the command to run this software on Linux OS.

```
tred.py tests/samples.csv --workdir work
```

Listing 3.3: Tredparse run - bash linux

In the end, I decid to choose this tool because among all existing RE callers, Tradparse is the most similar to ExpansionHunter and I use this tool to check ExpansionHunter results. Moreover, Tredparse models the paired-end distance which helps to improve accuracy, as shown in its reference paper [16], and also it has an added benefit of being able to compute the joint likelihood of the calls.

In Section 5.1, the choice of parameters is explained in detail.

3.2 Datasets

First of all, I tried to find some useful datasets in order to test each tool and then the whole pipeline. I tried to obtain some datasets mentioned in the papers in which they analyzed the various tools, from the EGA archive. This is in order to have a large pool of samples to test the number of true/false positives/negatives and also to already have results relating to those datasets to be compared. These datasets are related to exome sequences of individuals positive or negative for diseases caused by Repeat Expansions.

[EGAD00001003512 [18]]: It includes BAM files from 58 samples. These BAM files include all read pairs where at least one of the reads aligned within 1kb of the HTT repeat expansion. These samples were sequenced using 2x150bp reads on an Illumina HiSeqX.

[EGAD00001003562 [19]]: It includes BAM files from 120 samples. These samples were sequenced using 2x150bp reads on an Illumina HiSeqX.

But these datasets are not open access.

In general, I need a dataset with N positive samples to target diseases and M negative samples. The samples may be genome and exome sequences in order to perform different types of tests. Moreover, the sequences may have good coverage and short reads.

Here, I schematically report the sequencing details of the real samples collected to create a dataset that can be used in the set-up and testing phases of the pipeline and individual tools. These samples refer to individuals.

SAMPLE	COVERAGE	READS LEN	GEN/EXO
Trio mother	120x	150bp	Exome
Trio father	120x	150bp	Exome
Trio proban	120x	150bp	Exome
Trio mother	40x	150bp	Genome
Trio father	40x	150bp	Genome
Trio proband	40x	150bp	Genome
HD	120x	150bp	Exome
SCA1	120x	150bp	Exome
SCA3	120x	150bp	Exome
SCA7	120x	150bp	Exome
SCA1	40x	150bp	Genome
SCA3	40x	150bp	Genome
SCA7	40x	150bp	Genome

Table 3.1: Dataset features

The DNA sample was randomly fragmented by Covaris technology and the size of the library fragments was mainly distributed between 150bp and 250bp. The end repair of DNA fragments was performed and an "A" base was added at the 3'- end of each strand. Adapters were then ligated to both ends of the end repaired/dA tailed DNA fragments for amplification and sequencing. Size-selected DNA fragments were amplified by ligation-mediated PCR (LM- PCR), purified, and hybridized to the exome array for enrichment. The rolling circle amplification (RCA) was performed to loaded on BGISEQ sequencing platforms, and we performed high-throughput sequencing for each captured library to ensure that each sample met the desired average sequencing coverage.

I collect some real samples to build a useful and heterogeneous dataset.

The dataset is composed of negative samples, w.t.r. Repeats Expansion diseases, and positive ones. The six negative samples, that they have not pathological RE, are a trio (father, mother and proband) both genome and exome sequences.

The other seven samples are positive with respect to the Huntington's disease, Ataxia Spinocerebellar types 1, 3 and 7. The SCAs samples (genomes and exomes) are used only for the validation phase of the whole pipeline because they were sequenced at a later time.

3.3 Target deseases

Theoretical research was required about diseases caused by Repeat Expansions. Following, I report the information about neurodegenerative diseases of interest, related regions/genes involved and range of pathological expansions [2].

On the following table, there are all diseases that the pipeline searches within genome or exome sequences. For each of them are reported:

- Motif: the pattern repeated in the specific region, which could lead to disease.
- Gene: the gene that corresponds to the reference locus in which to search the expansion of each pathology.
- Locus: the position in which to search the expansion (chromosome and relative position in it). Due to some difference in the reference region among the various papers, during the analysis I use an error range of $\pm$ 100bp with respect to the indicated coordinates.
- Normal range: it's the range of the number of repeats for which the individual is negative for the relative disease.
- Premunition range: it's the interval of values that indicates the number of repetitions that are not pathological, but with high probability in the next generation it could become so.
- Pathological range: it's the range of values of the number of repeats for which the subject is positive for the relative disease.
- Type: it specifics if the disease is exonic or intronic, recessive or dominant. This indicates that if the disease is intronic it is necessary to consider the complete genome. Furthermore, it is necessary to analyze the presence of triplets in both alleles as if the disease is recessive to determine whether the individual is sick or not, both must be pathological.

DISEASE	MOTIF	GENE	LOCUS	NORMAL	PREMUNITION	PATHOLOGICAL	TYPE
SCA1	CAG	ATXN1	6p22.3	6 - 36	37 - 43	44 - 83	D - E
SCA2	CAG	ATXN2	12q24.12	15 - 31		34 - 220	D - E
SCA3	CAG	ATXN3	14q24.3-q31	12 - 40	41 - 53	54 - 86	D - E
SCA6	CAG	CACNA1A	19p13.2	4 - 18		21 - 33	D - E
SCA7	CAG	ATXN7	3p14.1	4 - 19	20 - 32	33 - 300	D - E
SCA8	CTA	ATXN8OS	13q21	15 - 50	51 - 70	71	D - E
SCA10	ATTCT	ATXN10	22q13.31	10 - 29	29 - 800	¿800	D - I
SCA12	CAG	PPP2R2B	5q32	7 - 31	32 - 54	55 - 78	D - E
SCA17	CAG	TBP	6q27	25 - 42		46 - 63	D - E
SCA36	CGCCTG	NOP56	20p13	5 - 13		13	D - I
SCA Friedreich	GAA	FXN	9q21.11	8 - 33	34 - 89	90	R - I
FXTAS	CGG	FMR1	Xq27.3	1 - 54	55 - 200	200	D - E
HD	CAG	HTT	4p16.3	27 - 34		34	D - E
SBMA	CAG	AR	Xq12	9 - 36		38 - 68	D - E
DRPLA	CAG	ATN1	12p13.31	3 - 35	36 - 47	48 - 93	D - E
OPMD	GCG	PABPN1	14q11.2	6 - 10		12 - 17	D - E
HDL2	CAG	JPH3	16q24.2	6 - 28	29 - 40	41 - 58	D - E
BPES	GCN	FOXL2	3q23	14 - 18		19 - 24	D - E
HPE5	GCN	ZIC2	13q32	15	16 - 24	25	D - E
MRGH	GCN	SOX3	Xq26.3	15	16 -25	26	D - E
CCHS	GCN	PHOX2B	4p13	20		25 - 29	D - E

Table 3.2: Neuro diseases. D=dominant, R=recessive, E=exonic, I=intronic

Chapter 4

Implementation

After software selection for the identification of the Repeat Expansion diseases to focus on, the design and implementation of the computational pipeline were done.

This method uses three different tools and the subsequent merge of the results. So, this pipeline is tested both on genome and exome sequences. In the following section, I describe the general workflow and the implementation details of the whole computational pipeline.

First of all, appropriate input files and parameters setting had to be created for the functioning of the Repeat Expansions identification tools. These files are about the repeats' database for each tool in order to find a specific list of diseases related to RE (e.g. STRs decoy for STRetch and variants catalog for ExpansionHunter). Next, a method was designed to utilize and merge the results from these pre-existing programmes.

To achieve the goal of this project, the developed computational pipeline is divided into the following main steps.

- Take in input the BAM file of the sample that will need to be analyzed and the types of the sequence (genome or exome). Then the program sorts and indexes this file using Samtools.

- Run independently ExpansionHunter, STRetch and Tredparse over the sample file. At the end, three different output files are produced with the REs identified from each tool.

- The useful information from each tool's output is extracted and saved in three specific data structures at the running time. Then, a comparison is performed in order to select the common RE. Firstly, the program selects all the RE from the three tools that are in the same regions (w.t.r. chromosomes and start/end of specific region). These regions are those identified as being of interest, ie in which to look for REs causing disease. The script uses the ExpansionHunter output as a reference for the regions, as it reports all specific regions (even if the RE number is 0) and therefore is a copy of the reference file used later. This is for simplicity of implementation of the comparison process. Then, the program performs another check w.t.r. common motif in the selected REs.

At the end of this phase, there is a new data structure in which the regions where RE have been detected in all tools are saved. In this structure are reported the number of REs for each allele of ExpansionHunter and Tredparse, the p-value on STRetch on this region, the related diseases/genes and the repeated pattern. It should be noted that the reported p-value indicates the accuracy of the reported values for that particular region, therefore it is used only as an evaluation index. For the total value of the number of repetitions the outputs of Tredparse and Expansion Hunter are used. Furthermore, in the superimposition of the regions an error range of + -100 is used as even in literature the values do not always coincide.

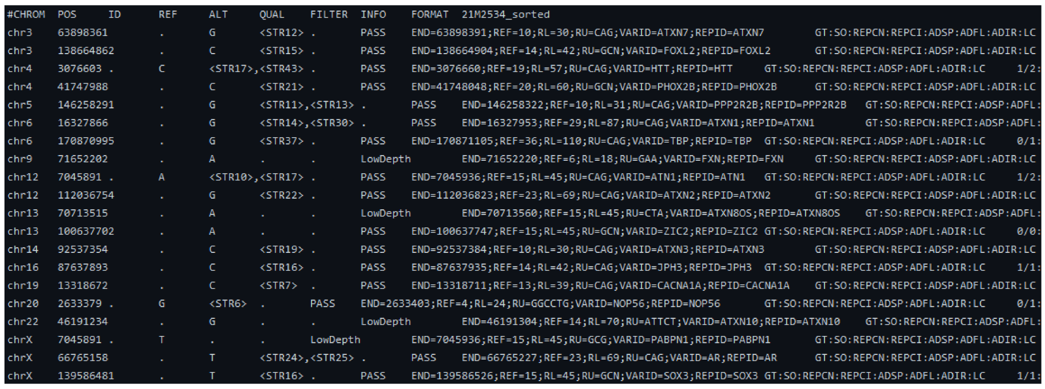
- The program computes the final values of REs for each allele as the average between ExpansionHunter and Tredparse. Considering that some of the diseases caused by repeat expansion are recessive, it is necessary to analyze the two alleles separately.

  First of all, the type of pathology (recessive or dominant) is checked for each RE reported from the previous filtering phase. Subsequently, for each REs the number of repetitions obtained is compared with the reference intervals (different for pathology). If the number of repetitions of both (recessive case) or at least one allele (dominant) falls within the alarm range then the sample is positive for that particular disease. Also, it is verified if one of the two alleles is instead in the premutation area. In the end, only the REs that are within one of these two reference intervals are saved.

- The final result is saved in two different output files for pathological and premutation cases.

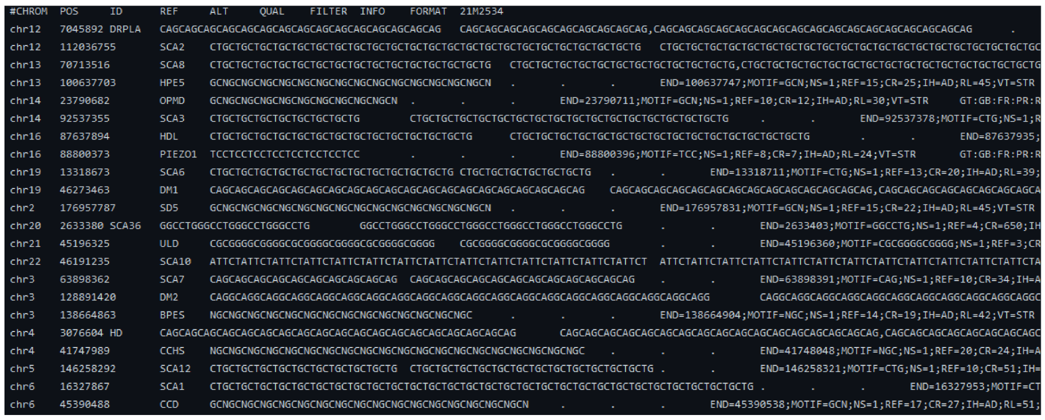
The differences between exome and genome analysis are only at the level of the single tools and in considering or not the intronic regions in the final comparison.

The following figure 4.1 shows an example of output for each tool used. Even though the tools use different approaches to counting repeats, they all output similar information.



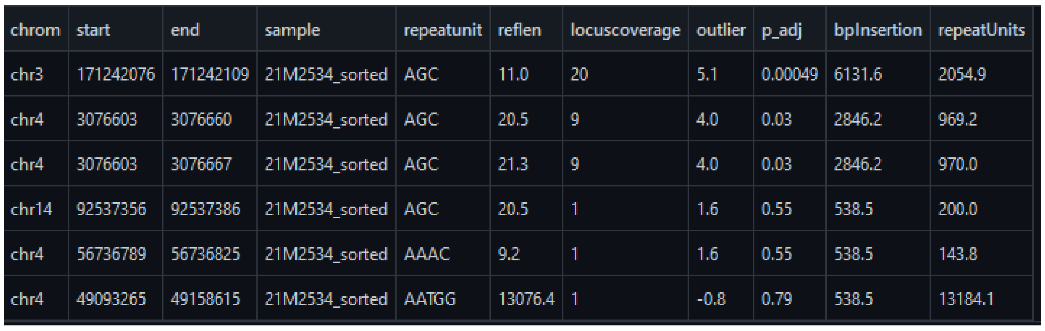
```
#CHROM  POS       ID      REF     ALT     QUAL    FILTER  INFO    FORMAT  21M2534_sorted
chr3    63898361          .       G       <STR12> .       PASS    END=63898391;REF=10;RL=30;RU=CAG;VARID=ATXN7;REPID=ATXN7        GT:SO:REPCN:REPCI:ADSP:ADFL:ADIR:LC
chr3    138664862         .       C       <STR15> .       PASS    END=138664904;REF=14;RL=42;RU=GCN;VARID=FOXL2;REPID=FOXL2       GT:SO:REPCN:REPCI:ADSP:ADFL:ADIR:LC
chr4    3076603 .         C       <STR17>,<STR43> .       PASS    END=3076660;REF=19;RL=57;RU=CAG;VARID=HTT;REPID=HTT     GT:SO:REPCN:REPCI:ADSP:ADFL:ADIR:LC     1/2:
chr4    41747988          .       C       <STR21> .       PASS    END=41748048;REF=20;RL=60;RU=GCN;VARID=PHOX2B;REPID=PHOX2B      GT:SO:REPCN:REPCI:ADSP:ADFL:ADIR:LC
chr5    146258291         .       G       <STR11>,<STR13> .       PASS    END=146258322;REF=10;RL=31;RU=CAG;VARID=PPP2R2B;REPID=PPP2R2B   GT:SO:REPCN:REPCI:ADSP:ADFL:
chr6    16327866          .       G       <STR14>,<STR30> .       PASS    END=16327953;REF=29;RL=87;RU=CAG;VARID=ATXN1;REPID=ATXN1        GT:SO:REPCN:REPCI:ADSP:ADFL:
chr6    170870995         .       G       <STR37> .       PASS    END=170871105;REF=36;RL=110;RU=CAG;VARID=TBP;REPID=TBP  GT:SO:REPCN:REPCI:ADSP:ADFL:ADIR:LC     0/1:
chr9    71652202          .       A       .       .       LowDepth        END=71652220;REF=6;RL=18;RU=GAA;VARID=FXN;REPID=FXN     GT:SO:REPCN:REPCI:ADSP:ADFL:ADIR:LC
chr12   7045891 .         A       <STR10>,<STR17> .       PASS    END=7045936;REF=15;RL=45;RU=CAG;VARID=ATN1;REPID=ATN1   GT:SO:REPCN:REPCI:ADSP:ADFL:ADIR:LC     1/2:
chr12   112036754         .       G       <STR22> .       PASS    END=112036823;REF=23;RL=69;RU=CAG;VARID=ATXN2;REPID=ATXN2       GT:SO:REPCN:REPCI:ADSP:ADFL:ADIR:LC
chr13   70713515          .       A       .       .       LowDepth        END=70713560;REF=15;RL=45;RU=CTA;VARID=ATXN8OS;REPID=ATXN8OS    GT:SO:REPCN:REPCI:ADSP:ADFL:
chr13   100637702         .       A       .       .       PASS    END=100637747;REF=15;RL=45;RU=GCN;VARID=ZIC2;REPID=ZIC2 GT:SO:REPCN:REPCI:ADSP:ADFL:ADIR:LC     0/0:
chr14   92537354          .       C       <STR19> .       PASS    END=92537384;REF=10;RL=30;RU=CAG;VARID=ATXN3;REPID=ATXN3        GT:SO:REPCN:REPCI:ADSP:ADFL:ADIR:LC
chr16   87637893          .       C       <STR16> .       PASS    END=87637935;REF=14;RL=42;RU=CAG;VARID=JPH3;REPID=JPH3  GT:SO:REPCN:REPCI:ADSP:ADFL:ADIR:LC     1/1:
chr19   13318672          .       C       <STR7>  .       PASS    END=13318711;REF=13;RL=39;RU=CAG;VARID=CACNA1A;REPID=CACNA1A    GT:SO:REPCN:REPCI:ADSP:ADFL:ADIR:LC
chr20   2633379 .         G       <STR6>  .       PASS    END=2633403;REF=4;RL=24;RU=GGCCTG;VARID=NOP56;REPID=NOP56       GT:SO:REPCN:REPCI:ADSP:ADFL:ADIR:LC     0/1:
chr22   46191234          .       G       .       .       LowDepth        END=46191304;REF=14;RL=70;RU=ATTCT;VARID=ATXN10;REPID=ATXN10    GT:SO:REPCN:REPCI:ADSP:ADFL:
chrX    7045891 .         T       .       .       LowDepth        END=7045936;REF=15;RL=45;RU=GCG;VARID=PABPN1;REPID=PABPN1       GT:SO:REPCN:REPCI:ADSP:ADFL:ADIR:LC
chrX    66765158          .       T       <STR24>,<STR25> .       PASS    END=66765227;REF=23;RL=69;RU=CAG;VARID=AR;REPID=AR      GT:SO:REPCN:REPCI:ADSP:ADFL:ADIR:LC
chrX    139586481         .       T       <STR16> .       PASS    END=139586526;REF=15;RL=45;RU=GCN;VARID=SOX3;REPID=SOX3 GT:SO:REPCN:REPCI:ADSP:ADFL:ADIR:LC     1/1:
```

(a) *Example of ExpansionHunter output file.*



```
#CHROM  POS       ID      REF     ALT     QUAL    FILTER  INFO    FORMAT  21M2534
chr12   7045892 DRPLA   CAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAG   CAGCAGCAGCAGCAGCAGCAGCAGCAGCAG,CAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAG      .
chr12   112036755       SCA2    CTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTG   CTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGC
chr13   70713516        SCA8    CTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTG   CTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTG,CTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTG
chr13   100637703       HPE5    GCNGCNGCNGCNGCNGCNGCNGCNGCNGCNGCNGCNGCNGCNGCN   .       .       .       END=100637747;MOTIF=GCN;NS=1;REF=15;CR=25;IH=AD;RL=45;VT=STR
chr14   23790682        OPMD    GCNGCNGCNGCNGCNGCNGCNGCNGCNGCN  .       .       .       END=23790711;MOTIF=GCN;NS=1;REF=10;CR=12;IH=AD;RL=30;VT=STR     GT:GB:FR:PR:R
chr14   92537355        SCA3    CTGCTGCTGCTGCTGCTGCTGCTG        CTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTG     .       .       END=92537378;MOTIF=CTG;NS=1;R
chr16   87637894        HDL     CTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTG      CTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTG        .       .       END=87637935;
chr16   88800373        PIEZO1  TCCTCCTCCTCCTCCTCCTCCTCC        .       .       .       END=88800396;MOTIF=TCC;NS=1;REF=8;CR=7;IH=AD;RL=24;VT=STR       GT:GB:FR:PR:R
chr19   13318673        SCA6    CTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTG CTGCTGCTGCTGCTGCTG    .       .       END=13318711;MOTIF=CTG;NS=1;REF=13;CR=20;IH=AD;RL=39;
chr19   46273463        DM1     CAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAG    CAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAG,CAGCAGCAGCAGCAGCAGCAGCAGCA
chr2    176957787       SD5     GCNGCNGCNGCNGCNGCNGCNGCNGCNGCNGCNGCNGCNGCNGCN   .       .       .       END=176957831;MOTIF=GCN;NS=1;REF=15;CR=22;IH=AD;RL=45;VT=STR
chr20   2633380 SCA36   GGCCTGGGCCTGGGCCTGGGCCTG        GGCCTGGGCCTGGGCCTGGGCCTGGGCCTGGGCCTGGGCCTG      .       .       END=2633403;MOTIF=GGCCTG;NS=1;REF=4;CR=650;IH
chr21   45196325        ULD     CGCGGGGCGGGGCGCGGGGCGGGGCGCGGGGCGGGG    CGCGGGGCGGGGCGCGGGGCGGGG        .       .       END=45196360;MOTIF=CGCGGGGCGGGG;NS=1;REF=3;CR
chr22   46191235        SCA10   ATTCTATTCTATTCTATTCTATTCTATTCTATTCTATTCTATTCTATTCTATTCTATTCTATTCTATTCT  ATTCTATTCTATTCTATTCTATTCTATTCTATTCTATTCTATTCTATTCTATTCTATTCTA
chr3    63898362        SCA7    CAGCAGCAGCAGCAGCAGCAGCAGCAGCAG  CAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAG    .       .       END=63898391;MOTIF=CAG;NS=1;REF=10;CR=34;IH=A
chr3    128891420       DM2     CAGGCAGGCAGGCAGGCAGGCAGGCAGGCAGGCAGGCAGGCAGGCAGGCAGGCAGGCAGGCAGGCAGGCAGGCAGGCAGG        CAGGCAGGCAGGCAGGCAGGCAGGCAGGCAGGCAGGCAGGCAGGC
chr3    138664863       BPES    NGCNGCNGCNGCNGCNGCNGCNGCNGCNGCNGCNGCNGCNGC      .       .       .       END=138664904;MOTIF=NGC;NS=1;REF=14;CR=19;IH=AD;RL=42;VT=STR
chr4    3076604 HD      CAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAG       CAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAGCAG,CAGCAGCAGCAGCAGCAGCAGCAGC
chr4    41747989        CCHS    NGCNGCNGCNGCNGCNGCNGCNGCNGCNGCNGCNGCNGCNGCNGCNGCNGCNGCNGCNGC    .       .       .       END=41748048;MOTIF=NGC;NS=1;REF=20;CR=24;IH=A
chr5    146258292       SCA12   CTGCTGCTGCTGCTGCTGCTGCTGCTGCTG  CTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTG .       .       END=146258321;MOTIF=CTG;NS=1;REF=10;CR=51;IH=
chr6    16327867        SCA1    CTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTGCTG .       .       .       END=16327953;MOTIF=CT
chr6    45390488        CCD     GCNGCNGCNGCNGCNGCNGCNGCNGCNGCNGCNGCNGCNGCNGCNGCNGCN     .       .       .       END=45390538;MOTIF=GCN;NS=1;REF=17;CR=27;IH=AD;RL=51;
```

(b) *Example of Tredparse output file.*



chrom	start	end	sample	repeatunit	reflen	locuscoverage	outlier	p_adj	bpInsertion	repeatUnits
chr3	171242076	171242109	21M2534_sorted	AGC	11.0	20	5.1	0.00049	6131.6	2054.9
chr4	3076603	3076660	21M2534_sorted	AGC	20.5	9	4.0	0.03	2846.2	969.2
chr4	3076603	3076667	21M2534_sorted	AGC	21.3	9	4.0	0.03	2846.2	970.0
chr14	92537356	92537386	21M2534_sorted	AGC	20.5	1	1.6	0.55	538.5	200.0
chr4	56736789	56736825	21M2534_sorted	AAAC	9.2	1	1.6	0.55	538.5	143.8
chr4	49093265	49158615	21M2534_sorted	AATGG	13076.4	1	-0.8	0.79	538.5	13184.1

(c) *Example of STRetch output file.*

Figure 4.1: Examples of tools' output files.

All the tools report for each RE some common information as the chromosome, the start and end positions of the regions in which they were detected and the gene corresponding to this area. In addition to this information, other useful data is reported in various fields.

In the case of ExpansionHunter, all the REs found in the input database are analyzed and reported, if an expansion is not present and therefore the number of repetitions is equal to that of the reference genome (reported in the *REF* parameter of the *INFO* field) is indicated with . in the *ALT* field. Therefore, in this file the fields of interest are: *ALT* which reports the value of the repeat expansion of each allele, *INFO.RU* with the repeated motif and *INFO.REPID* which reports the corresponding gene.

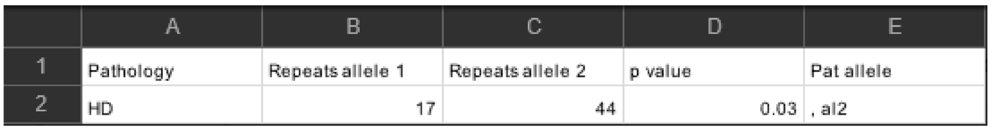
Instead, in the case of Tredparse, only the detected expansions are reported. The fields of interest are: *ID* in which the associated pathology is reported, *ALT* in which all the expansion is present, *INFO.MOTIF* which identifies the pattern of repeated bases and *INFO.REF* in which the value of the REs can be found for each allele in the format *REF=val_allele1, val_allele2.*

In the latter case, the information reported by STRetch is slightly different. In detail, in addition to the common ones, it reports the pattern adduced information(*repeatUnit*), the number of repeated units in the reference (*refLen*), the number of STR readings assigned to that locus (*locusCoverage*), z score testing for outliers (*outliers*), adjusted p-value that measures if this locus is expanded significantly compared to other samples (*p_adj*), the estimated allele size in bp inserted compared to the reference (*bpInsertions*) and the estimated total allele size in repeated units (*repeatUnits*). Due to the different information reported, the result produced by this tool is used only as a comparative index of the goodness of the result obtained in a certain area (p value).

From the intermediate outputs, the final result is reported in two files like the one below.

The number of repetitions for each RE extracted are compared with two different intervals, different for each specific disease. The first is an alarm interval that identifies the values for which REs become pathological (in this case it is necessary to analyze recessive and dominant cases separately). The second concerns the values for which REs are not pathological, but could become so in future generations of the individual. In this case, the two alleles are healed independently.

Therefore, the final outputs of the pipeline are two files (for each sample) different for the two analyzed ranges. As shown in the figure 4.2, in each one is reported the pathology, the number of repetitions for each allele and the associated pvalue. In both cases, in the last field, the allele (or alleles) that have a pathological or premutation value for that specific gene is also highlighted.

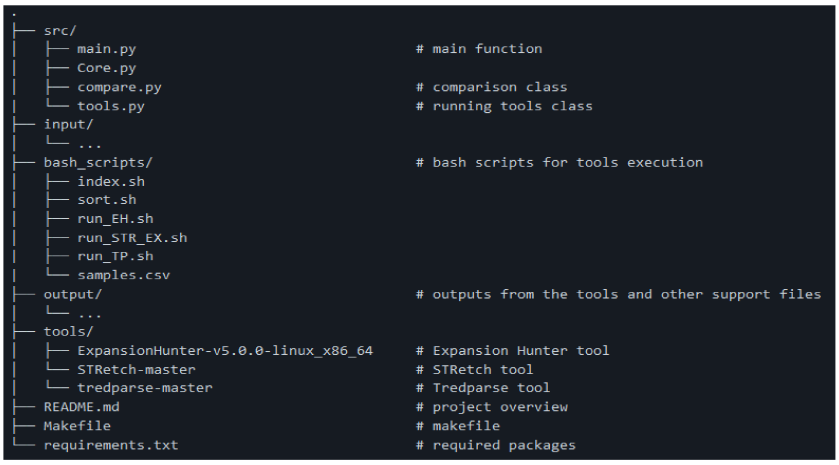


	A	B	C	D	E
1	Pathology	Repeats allele 1	Repeats allele 2	p value	Pat allele
2	HD	17	44	0.03	, al2

Figure 4.2: Example of whole pipeline's output file.

After this briefly introduction, some implementation aspects and the pipeline's organization are explained in more detail.

First of all, the project is divided into five main folders (*src*, *input*, *bash_scripts*, *output* and *tools*) in order to develop and optimally manage the entire structure. In the following image 4.3, the general structure of the project from its Github documentation is reported in order to clarify the idea.



```
.
├── src/
│   ├── main.py                               # main function
│   ├── Core.py
│   ├── compare.py                            # comparison class
│   └── tools.py                              # running tools class
├── input/
│   └── ...
├── bash_scripts/                             # bash scripts for tools execution
│   ├── index.sh
│   ├── sort.sh
│   ├── run_EH.sh
│   ├── run_STR_EX.sh
│   ├── run_TP.sh
│   └── samples.csv
├── output/                                   # outputs from the tools and other support files
│   └── ...
├── tools/
│   ├── ExpansionHunter-v5.0.0-linux_x86_64   # Expansion Hunter tool
│   └── STRetch-master                        # STRetch tool
│   └── tredparse-master                      # Tredparse tool
├── README.md                                 # project overview
├── Makefile                                  # makefile
└── requirements.txt                          # required packages
```

Figure 4.3: Project's stucture.

Specifically, the *src* folder contains all the implemented scripts which are used to analyze the input sample and subsequently compare the outputs of the various tools. To make it easier to read, the code has been divided into classes since the structure of the project, from a logical point of view, suggested a modular organization. Specifically, three distinct classes have been implemented: Core, Tools and Compare. Also, some functions could use one or more bash scripts.

Core

This class defines the execution structure of the entire pipeline. It provides the necessary functions to invoke the execution of the tools and subsequently the functions for the comparison of the results and the final output. First of all, the main function of this class, which is called in the *main* function, gives us a simple user interface for entering the necessary parameters and paths. So, after an input check, the *execute_tools* and *execute_comparison* functions are called. Respectively, they are the main functions for the running tools and results comparison tasks.

Tools

This class takes care of the first task of the pipeline. In detail, once an instance is initialized, the workflow is the following.

- Run the *sort.sh* bash script using the *subprocess.call* Python library function. This script performs the Samtools sorting of the sample BAM file. This task (together the indexing) is useful for the optimization of the tools' execution. The sample path is gaven in input to the script by $ reference.

- Run the *index.sh* bash script as the step above.

- Run ExpansionHunter tool with a specific bash script *run_EX.sh*. The sample name, project path and sample sequence type are provided as bash input directly from the script calling by Python as before. The bash script needs some input files for the tool's execution, they are in the *tools* folder.

- Run Tredparse tool by a specific bash script *run_TP.sh*. The sample name, project path and sample sequence type are provided as bash input directly from the script calling by Python. The bash script needs a file with the list of the sample. It is written by Python in this function.

- Run STRetch tool by a specific bash script *run_STR_EX.sh*. It exploits the exome pipeline for both exome and genome sequences. In the exome case, the bash script uses a BED file with the exonic regions specified. The sample name, project path and sample sequence type are provided as bash input directly from the script calling by Python (as before).

Compare

This class takes care of the last task in the pipeline. In detail, once an instance is initialized, the workflow is the following.

- Import the output files from the tools execution by the functions *import_files_EH* (.vcf), *import_files_tred_csv* (.csv and then .vcf), *import_files_str*

(.csv). In the tredparse case, the file is imported as .csv in order to modify some delimiters in the *INFO.RPA* field because the split function of Python uses ':' about saving values. It replaces ',' with ':'.

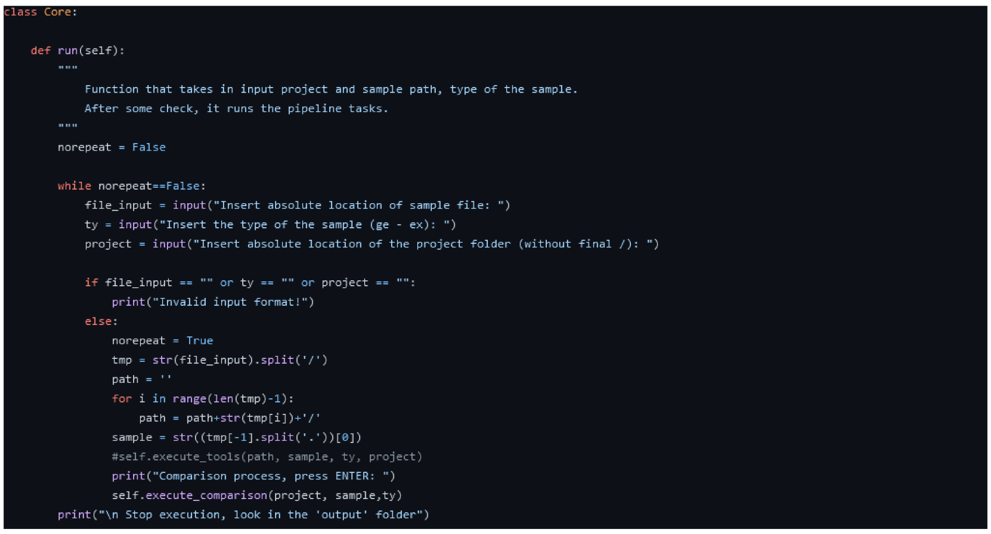
- Parse each file in order to extract and structure the useful information (e.g. chr, motif, start/end region, repeats value for each allele...). This task is performed by the *extract_eh*, *extract_tred* and *extract_stretch* functions.

- Compare RE regions from each tool's output by *comparison_regions* function. The program extracts the REs in the common regions looking at chromosome and their start/end region positions (add error interval of 100 bases).

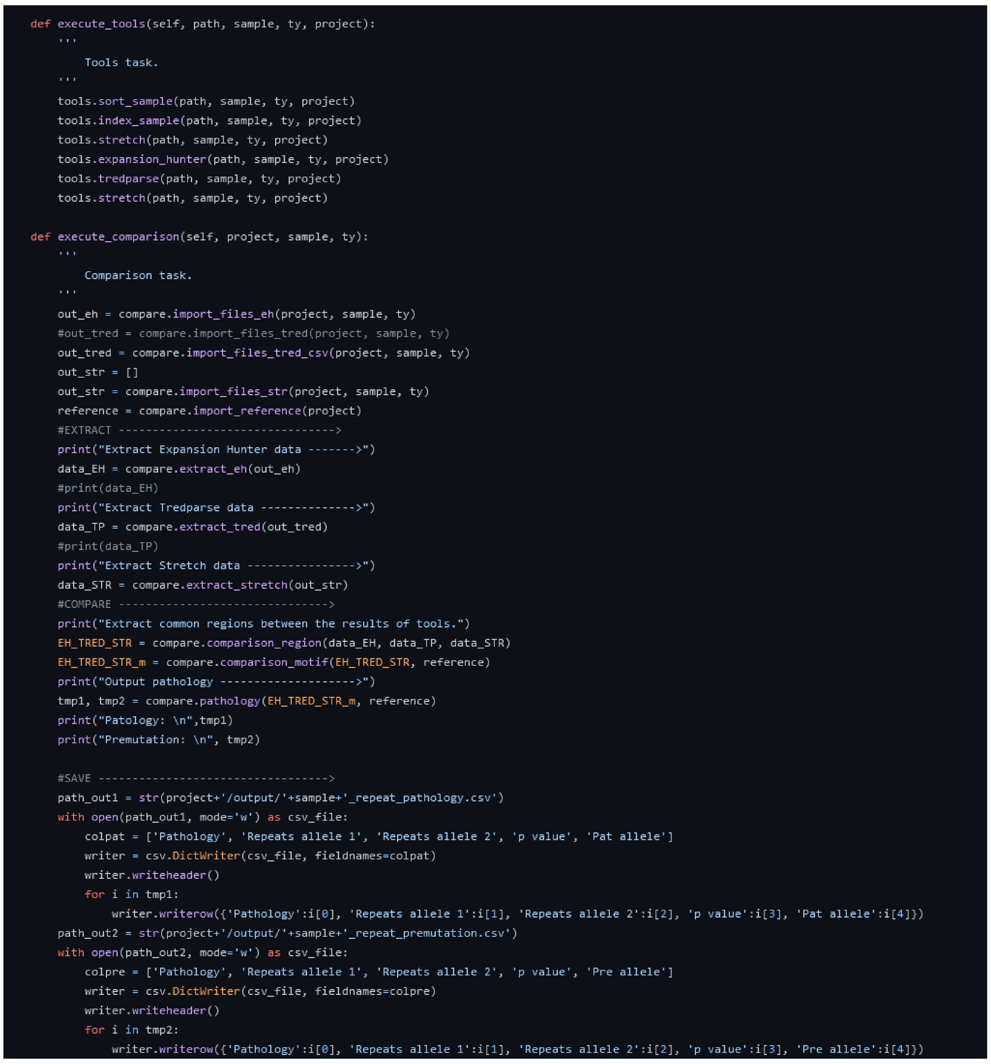
- Function *comparison_motif* filters the RE extracted before by motif. It saves RE only if the motif matches among tools (w.r.t. same regions) and with the reference.

- Function *pathology* compares each RE of the final subset with the reference pathological/premutation range for each allele. In this case, the program considers independently the alleles in the case of recessive or dominant diseases. It computes for each RE and alleles the values of repeats as the average between Expansion Hunter and Tredparse values. If the final repeat values are in its reference range, then they are reported into the specific output files.

In the figures 4.4, 4.5, 4.6 are reported the codes about Core, Tools and Compare classes, respectively.

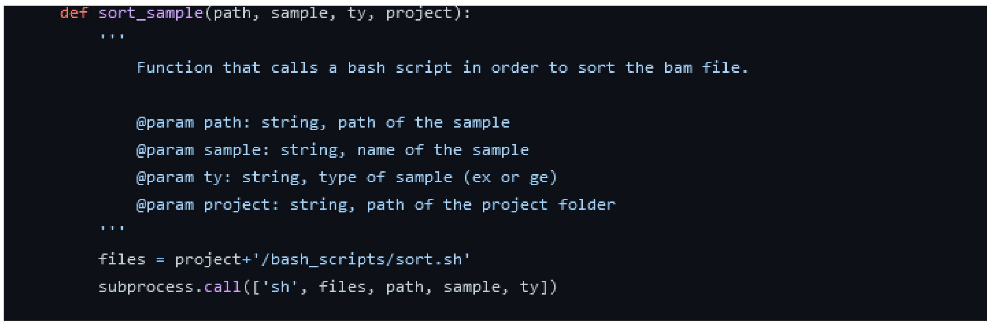


```python
class Core:

    def run(self):
        """
            Function that takes in input project and sample path, type of the sample.
            After some check, it runs the pipeline tasks.
        """
        norepeat = False

        while norepeat==False:
            file_input = input("Insert absolute location of sample file: ")
            ty = input("Insert the type of the sample (ge - ex): ")
            project = input("Insert absolute location of the project folder (without final /): ")

            if file_input == "" or ty == "" or project == "":
                print("Invalid input format!")
            else:
                norepeat = True
                tmp = str(file_input).split('/')
                path = ''
                for i in range(len(tmp)-1):
                    path = path+str(tmp[i])+'/'
                sample = str((tmp[-1].split('.'))[0])
                #self.execute_tools(path, sample, ty, project)
                print("Comparison process, press ENTER: ")
                self.execute_comparison(project, sample,ty)
        print("\n Stop execution, look in the 'output' folder")
```

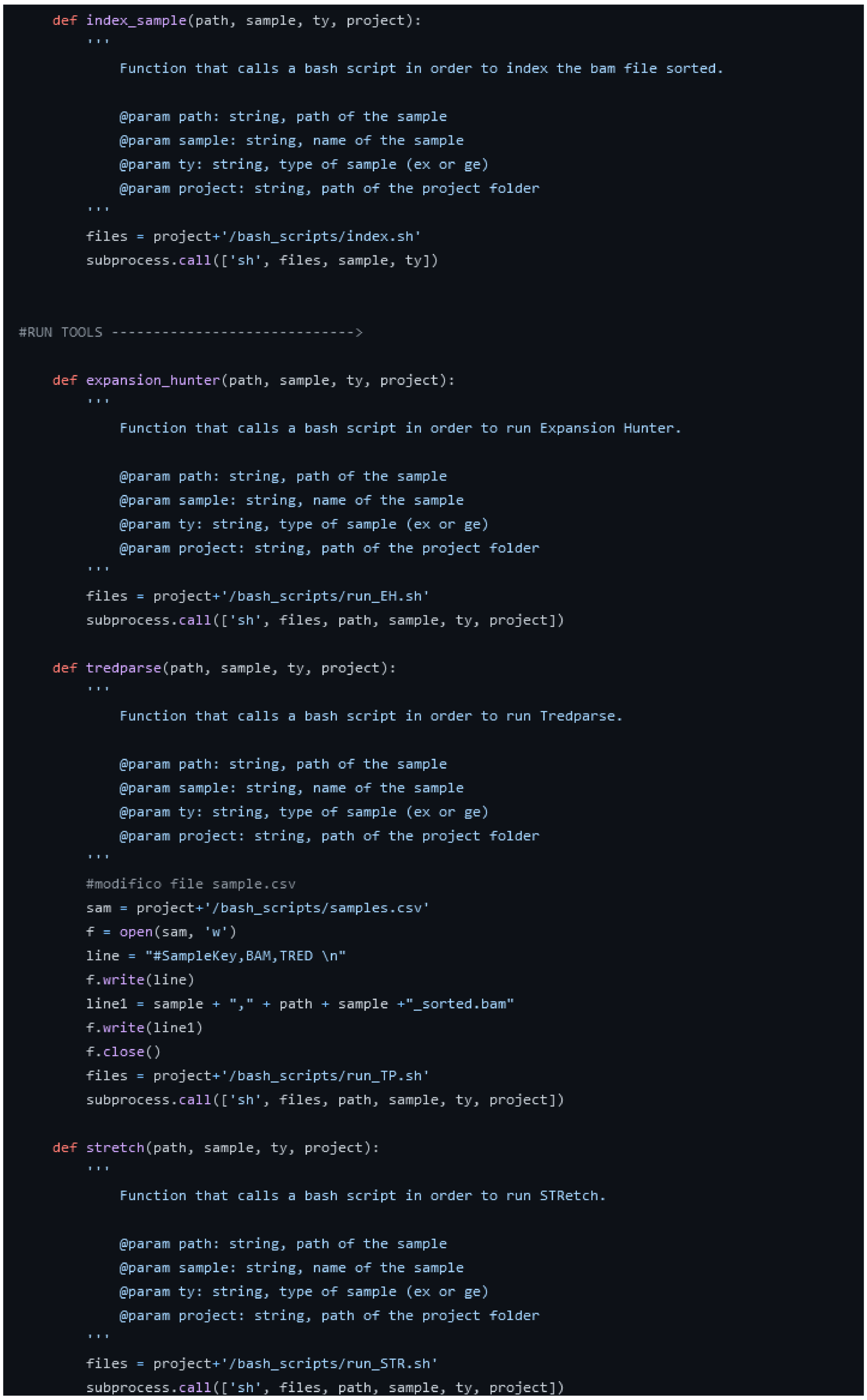


```
def execute_tools(self, path, sample, ty, project):
    '''
        Tools task.
    '''
    tools.sort_sample(path, sample, ty, project)
    tools.index_sample(path, sample, ty, project)
    tools.stretch(path, sample, ty, project)
    tools.expansion_hunter(path, sample, ty, project)
    tools.tredparse(path, sample, ty, project)
    tools.stretch(path, sample, ty, project)

def execute_comparison(self, project, sample, ty):
    '''
        Comparison task.
    '''
    out_eh = compare.import_files_eh(project, sample, ty)
    #out_tred = compare.import_files_tred(project, sample, ty)
    out_tred = compare.import_files_tred_csv(project, sample, ty)
    out_str = []
    out_str = compare.import_files_str(project, sample, ty)
    reference = compare.import_reference(project)
    #EXTRACT -------------------------------->
    print("Extract Expansion Hunter data ------->")
    data_EH = compare.extract_eh(out_eh)
    #print(data_EH)
    print("Extract Tredparse data -------------->")
    data_TP = compare.extract_tred(out_tred)
    #print(data_TP)
    print("Extract Stretch data ---------------->")
    data_STR = compare.extract_stretch(out_str)
    #COMPARE ------------------------------->
    print("Extract common regions between the results of tools.")
    EH_TRED_STR = compare.comparison_region(data_EH, data_TP, data_STR)
    EH_TRED_STR_m = compare.comparison_motif(EH_TRED_STR, reference)
    print("Output pathology ------------------->")
    tmp1, tmp2 = compare.pathology(EH_TRED_STR_m, reference)
    print("Patology: \n",tmp1)
    print("Premutation: \n", tmp2)

    #SAVE ---------------------------------->
    path_out1 = str(project+'/output/'+sample+'_repeat_pathology.csv')
    with open(path_out1, mode='w') as csv_file:
        colpat = ['Pathology', 'Repeats allele 1', 'Repeats allele 2', 'p value', 'Pat allele']
        writer = csv.DictWriter(csv_file, fieldnames=colpat)
        writer.writeheader()
        for i in tmp1:
            writer.writerow({'Pathology':i[0], 'Repeats allele 1':i[1], 'Repeats allele 2':i[2], 'p value':i[3], 'Pat allele':i[4]})
    path_out2 = str(project+'/output/'+sample+'_repeat_premutation.csv')
    with open(path_out2, mode='w') as csv_file:
        colpre = ['Pathology', 'Repeats allele 1', 'Repeats allele 2', 'p value', 'Pre allele']
        writer = csv.DictWriter(csv_file, fieldnames=colpre)
        writer.writeheader()
        for i in tmp2:
            writer.writerow({'Pathology':i[0], 'Repeats allele 1':i[1], 'Repeats allele 2':i[2], 'p value':i[3], 'Pre allele':i[4]})
```

Figure 4.4: Core script: it define the whole pipeline structure.

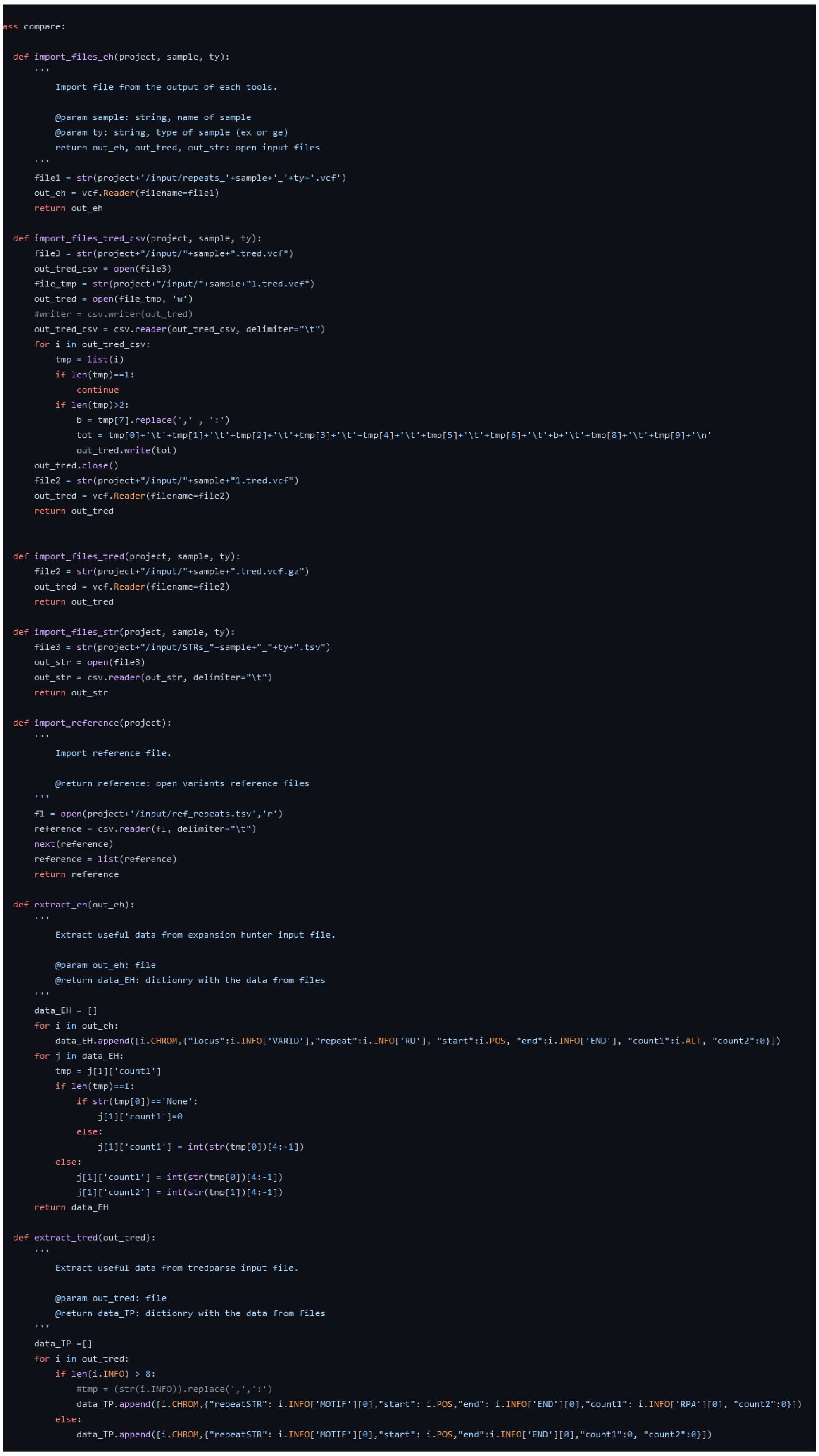


```
def sort_sample(path, sample, ty, project):
    '''
        Function that calls a bash script in order to sort the bam file.

        @param path: string, path of the sample
        @param sample: string, name of the sample
        @param ty: string, type of sample (ex or ge)
        @param project: string, path of the project folder
    '''
    files = project+'/bash_scripts/sort.sh'
    subprocess.call(['sh', files, path, sample, ty])
```

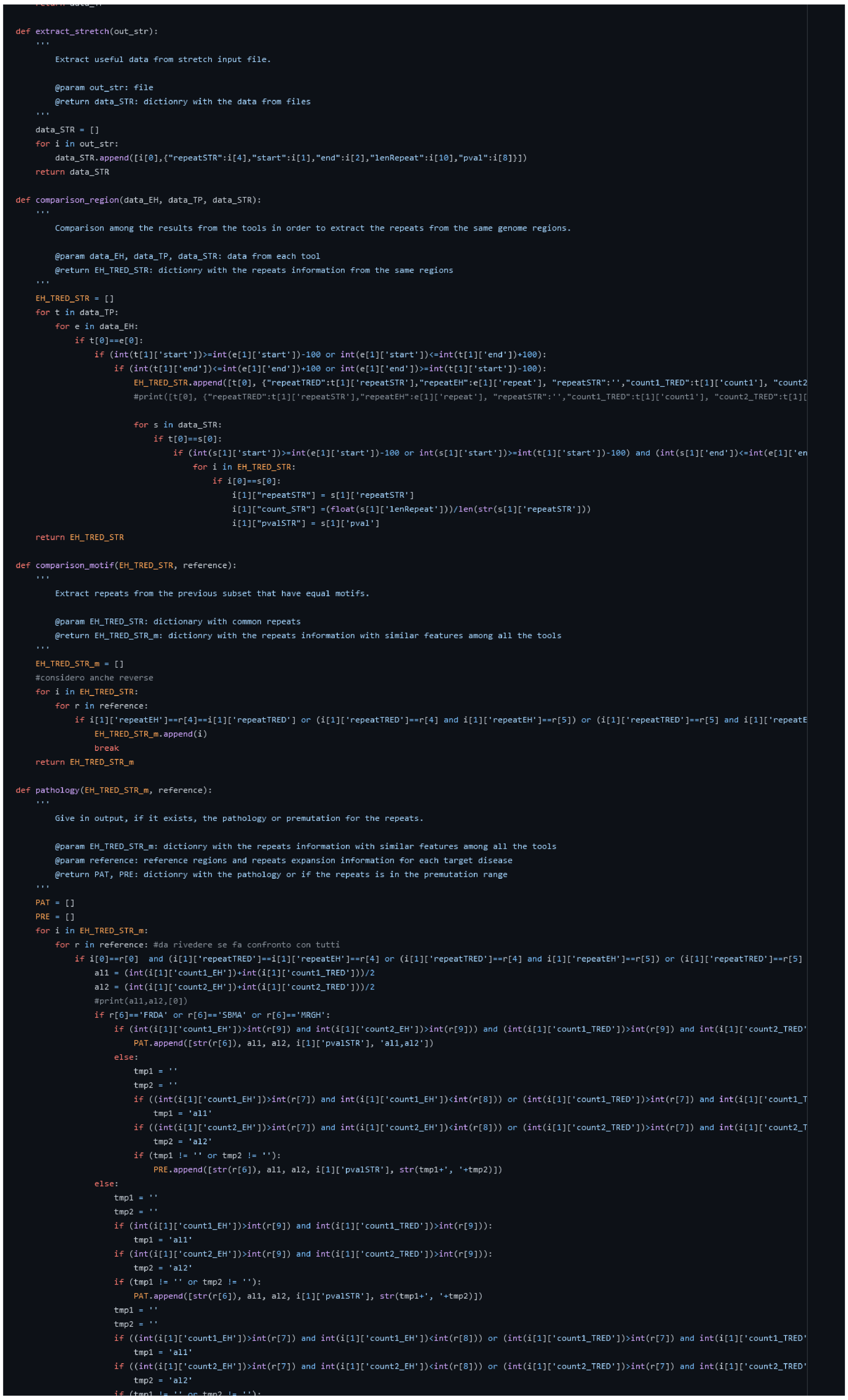


```
    def index_sample(path, sample, ty, project):
        '''
            Function that calls a bash script in order to index the bam file sorted.

            @param path: string, path of the sample
            @param sample: string, name of the sample
            @param ty: string, type of sample (ex or ge)
            @param project: string, path of the project folder
        '''
        files = project+'/bash_scripts/index.sh'
        subprocess.call(['sh', files, sample, ty])


#RUN TOOLS ---------------------------->

    def expansion_hunter(path, sample, ty, project):
        '''
            Function that calls a bash script in order to run Expansion Hunter.

            @param path: string, path of the sample
            @param sample: string, name of the sample
            @param ty: string, type of sample (ex or ge)
            @param project: string, path of the project folder
        '''
        files = project+'/bash_scripts/run_EH.sh'
        subprocess.call(['sh', files, path, sample, ty, project])

    def tredparse(path, sample, ty, project):
        '''
            Function that calls a bash script in order to run Tredparse.

            @param path: string, path of the sample
            @param sample: string, name of the sample
            @param ty: string, type of sample (ex or ge)
            @param project: string, path of the project folder
        '''
        #modifico file sample.csv
        sam = project+'/bash_scripts/samples.csv'
        f = open(sam, 'w')
        line = "#SampleKey,BAM,TRED \n"
        f.write(line)
        line1 = sample + "," + path + sample +"_sorted.bam"
        f.write(line1)
        f.close()
        files = project+'/bash_scripts/run_TP.sh'
        subprocess.call(['sh', files, path, sample, ty, project])

    def stretch(path, sample, ty, project):
        '''
            Function that calls a bash script in order to run STRetch.

            @param path: string, path of the sample
            @param sample: string, name of the sample
            @param ty: string, type of sample (ex or ge)
            @param project: string, path of the project folder
        '''
        files = project+'/bash_scripts/run_STR.sh'
        subprocess.call(['sh', files, path, sample, ty, project])
```

Figure 4.5: Tool script: it provide functions to execute all the tools and utility commands.



```
ass compare:

  def import_files_eh(project, sample, ty):
      '''
          Import file from the output of each tools.

          @param sample: string, name of sample
          @param ty: string, type of sample (ex or ge)
          return out_eh, out_tred, out_str: open input files
      '''
      file1 = str(project+'/input/repeats_'+sample+'_'+ty+'.vcf')
      out_eh = vcf.Reader(filename=file1)
      return out_eh

  def import_files_tred_csv(project, sample, ty):
      file3 = str(project+"/input/"+sample+".tred.vcf")
      out_tred_csv = open(file3)
      file_tmp = str(project+"/input/"+sample+"1.tred.vcf")
      out_tred = open(file_tmp, 'w')
      #writer = csv.writer(out_tred)
      out_tred_csv = csv.reader(out_tred_csv, delimiter="\t")
      for i in out_tred_csv:
          tmp = list(i)
          if len(tmp)==1:
              continue
          if len(tmp)>2:
              b = tmp[7].replace(',' , ':')
              tot = tmp[0]+'\t'+tmp[1]+'\t'+tmp[2]+'\t'+tmp[3]+'\t'+tmp[4]+'\t'+tmp[5]+'\t'+tmp[6]+'\t'+b+'\t'+tmp[8]+'\t'+tmp[9]+'\n'
              out_tred.write(tot)
      out_tred.close()
      file2 = str(project+"/input/"+sample+"1.tred.vcf")
      out_tred = vcf.Reader(filename=file2)
      return out_tred


  def import_files_tred(project, sample, ty):
      file2 = str(project+"/input/"+sample+".tred.vcf.gz")
      out_tred = vcf.Reader(filename=file2)
      return out_tred

  def import_files_str(project, sample, ty):
      file3 = str(project+"/input/STRs_"+sample+"_"+ty+".tsv")
      out_str = open(file3)
      out_str = csv.reader(out_str, delimiter="\t")
      return out_str

  def import_reference(project):
      '''
          Import reference file.

          @return reference: open variants reference files
      '''
      fl = open(project+'/input/ref_repeats.tsv','r')
      reference = csv.reader(fl, delimiter="\t")
      next(reference)
      reference = list(reference)
      return reference

  def extract_eh(out_eh):
      '''
          Extract useful data from expansion hunter input file.

          @param out_eh: file
          @return data_EH: dictionry with the data from files
      '''
      data_EH = []
      for i in out_eh:
          data_EH.append([i.CHROM,{"locus":i.INFO['VARID'],"repeat":i.INFO['RU'], "start":i.POS, "end":i.INFO['END'], "count1":i.ALT, "count2":0}])
      for j in data_EH:
          tmp = j[1]['count1']
          if len(tmp)==1:
              if str(tmp[0])=='None':
                  j[1]['count1']=0
              else:
                  j[1]['count1'] = int(str(tmp[0])[4:-1])
          else:
              j[1]['count1'] = int(str(tmp[0])[4:-1])
              j[1]['count2'] = int(str(tmp[1])[4:-1])
      return data_EH

  def extract_tred(out_tred):
      '''
          Extract useful data from tredparse input file.

          @param out_tred: file
          @return data_TP: dictionry with the data from files
      '''
      data_TP =[]
      for i in out_tred:
          if len(i.INFO) > 8:
              #tmp = (str(i.INFO)).replace(',',':')
              data_TP.append([i.CHROM,{"repeatSTR": i.INFO['MOTIF'][0],"start": i.POS,"end": i.INFO['END'][0],"count1": i.INFO['RPA'][0], "count2":0}])
          else:
              data_TP.append([i.CHROM,{"repeatSTR": i.INFO['MOTIF'][0],"start": i.POS,"end":i.INFO['END'][0],"count1":0, "count2":0}])
```



```
def extract_stretch(out_str):
    '''
        Extract useful data from stretch input file.

        @param out_str: file
        @return data_STR: dictionry with the data from files
    '''
    data_STR = []
    for i in out_str:
        data_STR.append([i[0],{"repeatSTR":i[4],"start":i[1],"end":i[2],"lenRepeat":i[10],"pval":i[8]}])
    return data_STR

def comparison_region(data_EH, data_TP, data_STR):
    '''
        Comparison among the results from the tools in order to extract the repeats from the same genome regions.

        @param data_EH, data_TP, data_STR: data from each tool
        @return EH_TRED_STR: dictionry with the repeats information from the same regions
    '''
    EH_TRED_STR = []
    for t in data_TP:
        for e in data_EH:
            if t[0]==e[0]:
                if (int(t[1]['start'])>=int(e[1]['start'])-100 or int(e[1]['start'])<=int(t[1]['end'])+100):
                    if (int(t[1]['end'])<=int(e[1]['end'])+100 or int(e[1]['end'])>=int(t[1]['start'])-100):
                        EH_TRED_STR.append([t[0], {"repeatTRED":t[1]['repeatSTR'],"repeatEH":e[1]['repeat'], "repeatSTR":'',"count1_TRED":t[1]['count1'], "count2
                        #print([t[0], {"repeatTRED":t[1]['repeatSTR'],"repeatEH":e[1]['repeat'], "repeatSTR":'',"count1_TRED":t[1]['count1'], "count2_TRED":t[1][

                        for s in data_STR:
                            if t[0]==s[0]:
                                if (int(s[1]['start'])>=int(e[1]['start'])-100 or int(s[1]['start'])>=int(t[1]['start'])-100) and (int(s[1]['end'])<=int(e[1]['en
                                    for i in EH_TRED_STR:
                                        if i[0]==s[0]:
                                            i[1]["repeatSTR"] = s[1]['repeatSTR']
                                            i[1]["count_STR"] =(float(s[1]['lenRepeat']))/len(str(s[1]['repeatSTR']))
                                            i[1]["pvalSTR"] = s[1]['pval']
    return EH_TRED_STR

def comparison_motif(EH_TRED_STR, reference):
    '''
        Extract repeats from the previous subset that have equal motifs.

        @param EH_TRED_STR: dictionary with common repeats
        @return EH_TRED_STR_m: dictionry with the repeats information with similar features among all the tools
    '''
    EH_TRED_STR_m = []
    #considero anche reverse
    for i in EH_TRED_STR:
        for r in reference:
            if i[1]['repeatEH']==r[4]==i[1]['repeatTRED'] or (i[1]['repeatTRED']==r[4] and i[1]['repeatEH']==r[5]) or (i[1]['repeatTRED']==r[5] and i[1]['repeatE
                EH_TRED_STR_m.append(i)
                break
    return EH_TRED_STR_m

def pathology(EH_TRED_STR_m, reference):
    '''
        Give in output, if it exists, the pathology or premutation for the repeats.

        @param EH_TRED_STR_m: dictionry with the repeats information with similar features among all the tools
        @param reference: reference regions and repeats expansion information for each target disease
        @return PAT, PRE: dictionry with the pathology or if the repeats is in the premutation range
    '''
    PAT = []
    PRE = []
    for i in EH_TRED_STR_m:
        for r in reference: #da rivedere se fa confronto con tutti
            if i[0]==r[0]  and (i[1]['repeatTRED']==i[1]['repeatEH']==r[4] or (i[1]['repeatTRED']==r[4] and i[1]['repeatEH']==r[5]) or (i[1]['repeatTRED']==r[5]
                al1 = (int(i[1]['count1_EH'])+int(i[1]['count1_TRED']))/2
                al2 = (int(i[1]['count2_EH'])+int(i[1]['count2_TRED']))/2
                #print(al1,al2,[0])
                if r[6]=='FRDA' or r[6]=='SBMA' or r[6]=='MRGH':
                    if (int(i[1]['count1_EH'])>int(r[9]) and int(i[1]['count2_EH'])>int(r[9])) and (int(i[1]['count1_TRED'])>int(r[9]) and int(i[1]['count2_TRED'
                        PAT.append([str(r[6]), al1, al2, i[1]['pvalSTR'], 'al1,al2'])
                    else:
                        tmp1 = ''
                        tmp2 = ''
                        if ((int(i[1]['count1_EH'])>int(r[7]) and int(i[1]['count1_EH'])<int(r[8])) or (int(i[1]['count1_TRED'])>int(r[7]) and int(i[1]['count1_T
                            tmp1 = 'al1'
                        if ((int(i[1]['count2_EH'])>int(r[7]) and int(i[1]['count2_EH'])<int(r[8])) or (int(i[1]['count2_TRED'])>int(r[7]) and int(i[1]['count2_T
                            tmp2 = 'al2'
                        if (tmp1 != '' or tmp2 != ''):
                            PRE.append([str(r[6]), al1, al2, i[1]['pvalSTR'], str(tmp1+', '+tmp2)])
                else:
                    tmp1 = ''
                    tmp2 = ''
                    if (int(i[1]['count1_EH'])>int(r[9]) and int(i[1]['count1_TRED'])>int(r[9])):
                        tmp1 = 'al1'
                    if (int(i[1]['count2_EH'])>int(r[9]) and int(i[1]['count2_TRED'])>int(r[9])):
                        tmp2 = 'al2'
                    if (tmp1 != '' or tmp2 != ''):
                        PAT.append([str(r[6]), al1, al2, i[1]['pvalSTR'], str(tmp1+', '+tmp2)])
                    tmp1 = ''
                    tmp2 = ''
                    if ((int(i[1]['count1_EH'])>int(r[7]) and int(i[1]['count1_EH'])<int(r[8])) or (int(i[1]['count1_TRED'])>int(r[7]) and int(i[1]['count1_TRED'
                        tmp1 = 'al1'
                    if ((int(i[1]['count2_EH'])>int(r[7]) and int(i[1]['count2_EH'])<int(r[8])) or (int(i[1]['count2_TRED'])>int(r[7]) and int(i[1]['count2_TRED'
                        tmp2 = 'al2'
                    if (tmp1 != '' or tmp2 != ''):
```

Figure 4.6: Compare script: it manage the comparison task among tools' outputs.

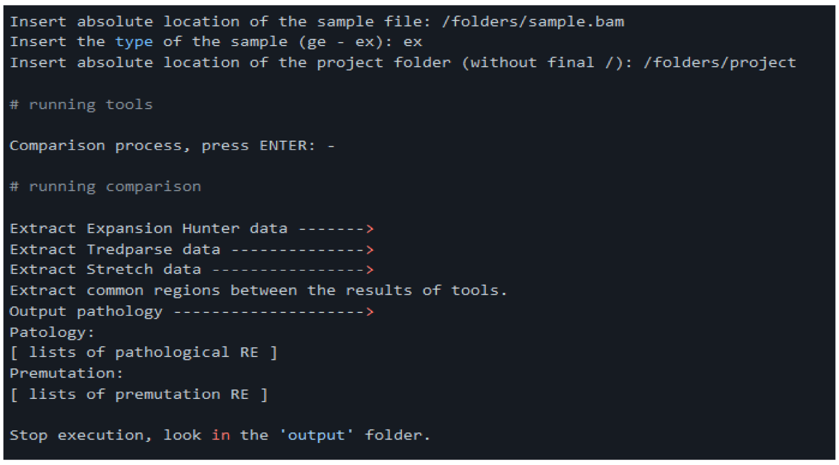
Then, the *bash_scripts* folder contains all the bash scripts needed for the tools execution or utility tasks. In particular, there is one script for each tool with the command for the execution and some command to rename or move the outputs in the correct folder. Also, there are other two scripts for sorting and indexing the bam file. Then, the *tools* folder holds the three tools' folder in a compressed version.

The last are the *input* folder, that contains all the input files from the tools that we intend to analyze, and the *output* where the output files from the whole pipeline are saved. The program gives in output two .csv files that contain the list of all the possible pathological REs and all the premutation REs, respectively.

In addition to the main folders, the *Makefile* and *requirements.txt* files are made available. Both of them can be useful to install automatically the minimum requirements and packages necessary for proper pipeline execution. The alternative is to install the Python libraries and used tools separately. The dependences and requirements are:

- Python with version greater or equal to 2.7.
- Python libraries: *os*, *subprocess*, *csv*, *sys*, *json*, *re*, *vcf*, *pandas*, *gzip*, *ast* and *pyrecord*.
- Expansion Hunter program: the compressed package is in the *tools* folder. It may be installed by *cmake* command. This type of installation required a *gcc* version greater or equal to 5.1.
- STRetch program: also in this case, the compressed package is in the *tools* folder. It required a Java version greater or equal to 1.8 and the program can be installed by *./install.sh* command.
- Tredparse is just in a executable version. The only thing is the unzip of the Tredparse folder.
- Samtools for the sorting and indexing tasks.

For example, an instance of the pipeline execution is shown in the figure 4.7. You can see in the first three lines the insert of the required inputs and then the various steps of the execution.



```
Insert absolute location of the sample file: /folders/sample.bam
Insert the type of the sample (ge - ex): ex
Insert absolute location of the project folder (without final /): /folders/project

# running tools

Comparison process, press ENTER: -

# running comparison

Extract Expansion Hunter data ------->
Extract Tredparse data -------------->
Extract Stretch data ---------------->
Extract common regions between the results of tools.
Output pathology -------------------->
Patology:
[ lists of pathological RE ]
Premutation:
[ lists of premutation RE ]

Stop execution, look in the 'output' folder.
```

Figure 4.7: Example of pipeline execution.

Chapter 5

Experimental results

In this chapter we show all the tests done. In particular, in Section 5.1 we report the parameters tunning and the relative trials. Then, we describe the results obtained in the tests with the real samples, both exome and genome sequences.

5.1 Parameters

An important part of the design phase is the tuning of the parameters for the three tools. For this task we used the part of the dataset composed of the negative samples for which there is no pathological expansion and the positive Huntington sample for which the number of repeated triplets in a specific region is known from the medical record (pathological on allele2 with 47 repeats).

The following tests were done to set up some parameters of the pipeline. Each selected tool was run with different parameters setting in order to choose the best one.

Initially, we started from Tredparse as it does not require the tuning of any parameters. The only option to be set in the input is the version of the reference genome with the parameter *–ref hg19* in this case. The samples selected from the test dataset were analyzed with Tredparse in order to obtain the output files with the counts of the triplets in the regions indicated in the database used (coinciding with those in the literature used as a reference by this pipeline).

Therefore, the Tredparse output was used as a reference point for the three negative samples and for the Huntington positive one, for which the values found for the repeat expansions correspond to those are reported in the medical record (Tredparse: allele1=17, allele2=45; pathological on allele2).

I analyze all the samples both exome and genome as it is necessary to find ad hoc parameters for both types of sequences (see Section 5.2).

Now, considering the other two tools, I report the tested parameters and the input files created specifically for their correct functioning in relation to the objectives of this pipeline. In addition to this, in this phase a reference file was also created with the intervals of pathological repeats and premutation, regions associated with the individual diseases and the relative triplets. This file is used at the point in the pipeline where the results obtained by the three tools are compared.

Expansion Hunter

The parameters of Expansion Hunter can be divided into two groups. The first one are the required parameters that are shown below.

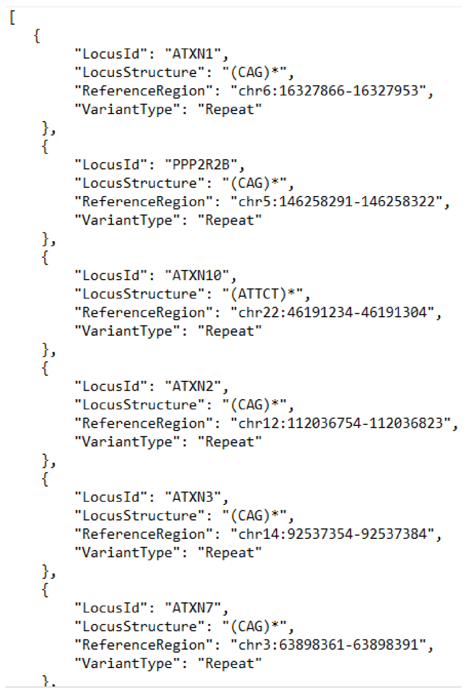
```
--reads <aligned reads BAM/CRAM file/URL>
--reference <reference genome FASTA file>
--variant-catalog <JSON file specifying variants to genotype>
--output-prefix <Prefix for the output files>
```

These parameters refer to the management of input and output files. The *reads* option takes the BAM or CRAM file of the sample which will be analyzed. In this case, I work only with a local BAM file (no by URL). *Reference* takes the reference genome in FASTA format with hg19 version. *variant-catalog* is used to define the path of the custom input file with the list of the only repeats that the program checks in the specified regions trying to identify them and counting the number of repetitions. It is a JSON file in which the repeats, that are related to the diseases of interest, are defined by: start/end position of the target region, the motif of repeats, name of the disease and name of the relative gene. An example of this file is shown in the figure 5.1.

The second group include the optional parameters:

```
--sex <m or f>
--min-locus-coverage <int>
--region-extension-length <bp>
--analysis-mode <seeking or streaming>
--threads <number of threads>
```

The first option *sex* defines if the sample refers to a male or female individual. The default is female. This parameter is only important for diseases affecting the X and Y chromosomes. In this project some diseases affecting repeat expansions on areas of these chromosomes (e.g. fragile X) are considered, so it is necessary to set it according to the sample being analyzed.



```
[
  {
      "LocusId": "ATXN1",
      "LocusStructure": "(CAG)*",
      "ReferenceRegion": "chr6:16327866-16327953",
      "VariantType": "Repeat"
   },
   {
      "LocusId": "PPP2R2B",
      "LocusStructure": "(CAG)*",
      "ReferenceRegion": "chr5:146258291-146258322",
      "VariantType": "Repeat"
   },
   {
      "LocusId": "ATXN10",
      "LocusStructure": "(ATTCT)*",
      "ReferenceRegion": "chr22:46191234-46191304",
      "VariantType": "Repeat"
   },
   {
      "LocusId": "ATXN2",
      "LocusStructure": "(CAG)*",
      "ReferenceRegion": "chr12:112036754-112036823",
      "VariantType": "Repeat"
   },
   {
      "LocusId": "ATXN3",
      "LocusStructure": "(CAG)*",
      "ReferenceRegion": "chr14:92537354-92537384",
      "VariantType": "Repeat"
   },
   {
      "LocusId": "ATXN7",
      "LocusStructure": "(CAG)*",
      "ReferenceRegion": "chr3:63898361-63898391",
      "VariantType": "Repeat"
   },
```

Figure 5.1: Part of the variant.json file.

The *min-locus-coverage* option specifies minimum read coverage depth at loci on diploid chromosomes required to attempt genotyping. Automatically reduced to half for loci on haploid chromosomes. The locus will be skipped if the coverage falls below this value. Some values were tested (e.g. 10, 20, 30) but it was preffered to leave the default value equal to 10 because the results doesn't change much.

Then, *region-extension-length* define how far from on/off target regions to search for informative reads (in bp). It is defined at 500bp because is the range used also in the script of the pipeline in which there is the comparison and the extraction of the common regions. However, it should be remembered that the regions that are analyzed are defined in the JSON file and they have different length but it is reasonable to define an error range.

The *analysis-mode* parameter specifies which analysis workflow to use, between seeking or streaming. In streaming mode, the alignment file is read in a single pass and all variants are analyzed during this reading operation. Streaming mode is recommended for the analysis of large catalogs, but does require more memory as a function of catalog size. But in this case seeking mode is recommended because there is a small repeats catalog, as specified in the Expansion Hunter documentation. In this mode, alignment file indexing is used to seek specific read sets for the analysis of each variant in a faster way. So, the sorting and indexing of the BAM file are required.

The last *threads* option defines the numbers of threads to can be used to accelerate the analysis of large variant catalogs. But, but considering that the repetition database is not very large, the default value (equal to 1) is sufficient. The analysis is however fast both on the exome and on the genome.

At the end, the final setting may be the following:

```
— —reads  sample_sorted.bam
— —reference  genome.fa
— —variant—catalog  variants.json
— —output—prefix  repeats_sample
— —region—extension—length  500
— —analysis—mode  seeking
```

So, after the execution of Expansion Hunter over all the selected samples, I can conclude that with these chosen parameters, the results close to the Tredparse one.

The parameters were tested mainly on exomic sequences as the most reliable real sample (Huntington) is exomic. As far as the analysis of genomic sequences is concerned, the chosen parameters still produce results in agreement with the reference ones obtained with Tredparse. The only trick is to insert regions and intronic repeats in the JSON file that Expansion Hunter uses as a reference database for searching for repeat expansions.

STRetch

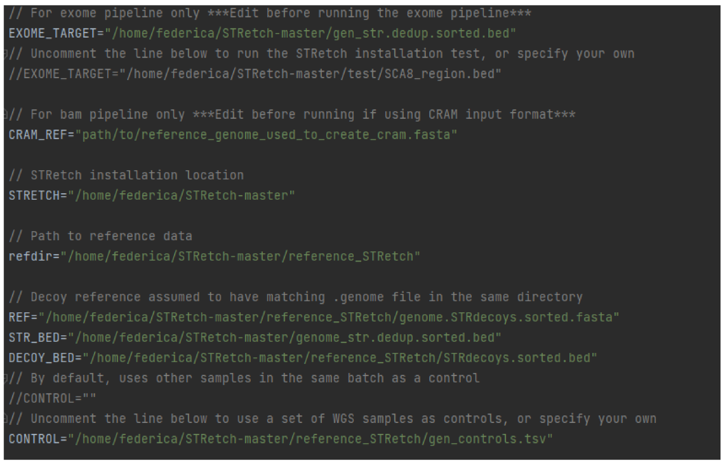
In general, the command to run STRetch has the following format. After the tests reported in Section 5.2, I use the WGS pipeline for the analysis of the BAM input file. In the case of exome sequences, it takes in input also the list of the exom regions.

```
tools/bin/bpipe run pipelines/STRetch_wg_bam_pipeline.groovy
    —p bwa_parallelism=<int> —p EXOME_TARGET="target_region.bed"
    sample.bam
```

This tools required the following parameters:

```
—p bwa_parallelism = <int>
—p EXOME_TARGET = "target_region.bed"
input_regions = "STR_list.bed"
```

The *bwa_parallelism* option allows us to run multiple instances of BWA in parallel to speed up the read extraction and mapping time. This option only affects the time complexity of the process, but not the goodness of the



```
// For exome pipeline only ***Edit before running the exome pipeline***
EXOME_TARGET="/home/federica/STRetch-master/gen_str.dedup.sorted.bed"
// Uncomment the line below to run the STRetch installation test, or specify your own
//EXOME_TARGET="/home/federica/STRetch-master/test/SCA8_region.bed"

// For bam pipeline only ***Edit before running if using CRAM input format***
CRAM_REF="path/to/reference_genome_used_to_create_cram.fasta"

// STRetch installation location
STRETCH="/home/federica/STRetch-master"

// Path to reference data
refdir="/home/federica/STRetch-master/reference_STRetch"

// Decoy reference assumed to have matching .genome file in the same directory
REF="/home/federica/STRetch-master/reference_STRetch/genome.STRdecoys.sorted.fasta"
STR_BED="/home/federica/STRetch-master/genome_str.dedup.sorted.bed"
DECOY_BED="/home/federica/STRetch-master/reference_STRetch/STRdecoys.sorted.bed"
// By default, uses other samples in the same batch as a control
//CONTROL=""
// Uncomment the line below to use a set of WGS samples as controls, or specify your own
CONTROL="/home/federica/STRetch-master/reference_STRetch/gen_controls.tsv"
```

Figure 5.2: Custom part of the pipeline_config.groovy file.

results. For instance, I try with 12 that is the value recommended in the documentation. If an exome sequence will be analyzed, the *EXOME_TARGET* takes a BED file that contains all the exome target regions.

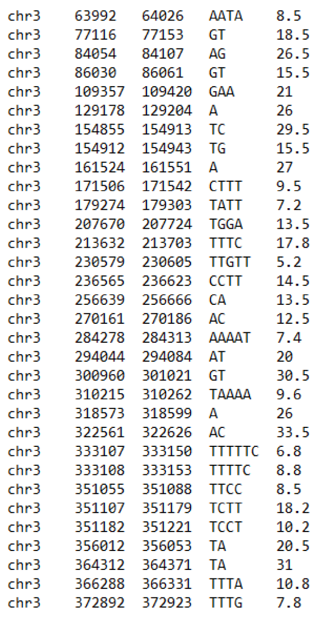
Furthermore, both in genome and exome case, it's necessary to edit the *pipeline_config.groovy* used by WGS or WES pipelines, as in figure 5.2.

This file defines the paths of the reference data and the used tools (e.g. bpipe). Also, it set the *EXOME_TARGET* parameter that imports the file with the list of every possible repeats for all exome regions. It is used only for the WES pipeline. The following figure reports an example of the file that defines these regions.

With the aim of making a more targeted analysis and reducing the computation time, we tried to filter the files with the lists of possible repetitions (with respect to the exomic regions and to the whole genome), keeping only the chromosomes affected by the diseases of study. The result is a lower number of repetitions detected, but the calculated values (e.g. p-value) are the same between the analysis cases with and without the filtered reference file.

The *REF* option refers to the fasta file of reference genome hg19 with the generated STR decoy chromosomes. Also, *STR_BED* and *DECOY_BED* options point to the bed files of the sorted repeats database for the genome. The indexes of this file must be in the same directory, but don't need to be specified. In order to generate STR decoy, it uses genome UCSC annotation for the STRs.

At the end, it defines the *CONTROL* file that is made from the exome son sample by the estimateSTR script. It is used as a control STR for the



```
chr3    63992   64026   AATA    8.5
chr3    77116   77153   GT      18.5
chr3    84054   84107   AG      26.5
chr3    86030   86061   GT      15.5
chr3    109357  109420  GAA     21
chr3    129178  129204  A       26
chr3    154855  154913  TC      29.5
chr3    154912  154943  TG      15.5
chr3    161524  161551  A       27
chr3    171506  171542  CTTT    9.5
chr3    179274  179303  TATT    7.2
chr3    207670  207724  TGGA    13.5
chr3    213632  213703  TTTC    17.8
chr3    230579  230605  TTGTT   5.2
chr3    236565  236623  CCTT    14.5
chr3    256639  256666  CA      13.5
chr3    270161  270186  AC      12.5
chr3    284278  284313  AAAAT   7.4
chr3    294044  294084  AT      20
chr3    300960  301021  GT      30.5
chr3    310215  310262  TAAAA   9.6
chr3    318573  318599  A       26
chr3    322561  322626  AC      33.5
chr3    333107  333150  TTTTTC  6.8
chr3    333108  333153  TTTTC   8.8
chr3    351055  351088  TTCC    8.5
chr3    351107  351179  TCTT    18.2
chr3    351182  351221  TCCT    10.2
chr3    356012  356053  TA      20.5
chr3    364312  364371  TA      31
chr3    366288  366331  TTTA    10.8
chr3    372892  372923  TTTG    7.8
```

Figure 5.3: Part of the exome_region_str.bed file.

analysis of the other samples.

The differences between the WGS and the WES pipelines are the files used as reference for the STRs decoy (BED file) and the definition of the target regions.

5.2 Results on real samples

Initially, some tests were performed to analyze the performance of the three tools and then set their parameters (see Section 5.1) also in relation to the study of both genomes and exomes. To carry out these initial analyses, the part of the dataset composed of the trio (child/mother/father samples) was used. After the end of the method implementation, it was tested using the remaining part of the dataset consisting of genomes and exomes positive for note REs diseases.

5.2.1 Genome and exome tests

Compared to the papers read ([9], [16] and [15]), all the tools chosen for this pipeline are used only with whole genome sequencing data. But it would be interesting to apply this computational pipeline also to exomic sequences because working with exomic data is much less expensive in terms of initial sequencing costs, data storage costs and computation time or costs. Most of the diseases of interest caused by Repeat Expansions are exomic. But it is necessary to take into account that some are intronic so sometimes it is necessary to analyze the entire genome to have a more complete clinical picture.

Expansion Hunter and Tredparse are WGS tools. Instead, STRetch uses different pipelines to be able to analyze both exome and genome, but with very different computational costs regarding the WES pipeline. Theoretically, it is interesting to be able to use the same pipeline on exomic sequences as precisely as for the genome.

With these tests we want to understand if it is reasonable to apply these tools and therefore also the complete pipeline to exomic sequences.

The part of the dataset composed of the three samples negative for RE diseases was considered, as they are both exomes and genomes for each sample. For these tests, positive samples are not necessary in which it is sufficient to compare the different outputs only to understand the common regions detected and the deviation in the number of repetitions detected.

Firstly, I run each tool independently on the three exomes and on the three genomes from the same sample. The parameters used in the various executions are those tested and reported in the next section. Subsequently, considering each tool individually, I compare the results for each sample. I extract the common regions and repeat counts between exome and genome. Always bearing in mind that some values will not be present in the exome sample outputs as they concern intronic regions.

Before reporting and discussing the analysis made on the results, we can make some considerations regarding the execution of the tools on the different samples. Looking at the computation times, it can be seen that exome analyzes are faster (about half the time) for all three tools. This is much more evident in STRetch because it is computationally more expensive as it checks for the presence of any type of repeated patterns. Due to the smaller size of the sample files, also with regard to the sample preparation phase (sorting and indexing of BAM files), the exome is more convenient.

However, it is necessary to check that the results are in any case compatible and precise as in the case of genomic analyzes.

Now, I report for each tool the analysis performed on one sample and some observations. For each tool, the files with the common repeats from the comparison of genome/exome results are shown w.t.r. each sample. Also in the tables, there are the number of the repeats identified in genome sequences, exome sequences and the common repeats.

Expansion Hunter

The values reported in table 5.1, are computed by subtracting from the total number of possible repetitions to be found (equal to 23, see file reference), the number of repetitions not found or with a count equal to 0. Instead, the number of common detected repeats was calculated by counting the repeats present in both the genome and exome with the same region and motif. This process was done thanks to a Python script that compares the two output files. The generated files are shown above.

Looking at the file showing in figure 5.4, the common values it can be seen that Expansion Hunter detects many common repetitions both in the genome and in the exome (see table) and the values are almost the same. Therefore, it can be said that this tool works well both genome and exomes, unless there are some uncertainty and intronic regions not detected.

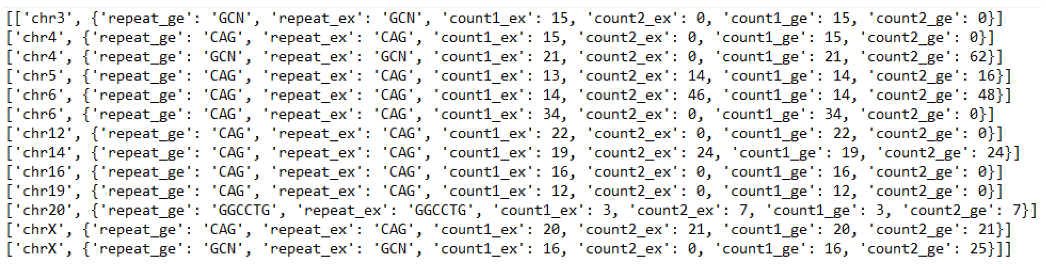
SAMPLE	GENOME	EXOME	COMMON
Son	20	14	13
Mother	19	17	15
Father	20	15	14

Table 5.1: Values of the repeat expansions identified by Expansion Hunter. They are respectively in genomes, exomes and then the number of the common repeats between them.

Tredparse

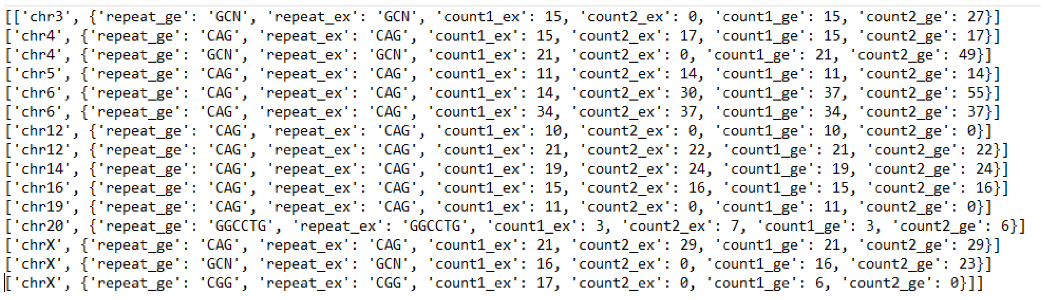
SAMPLE	GENOME	EXOME	COMMON
Mother	20	21	20
Father	21	20	17
Proband	19	18	18

Table 5.2: Values of the repeat expansions identified by Tredparse. They are respectively in genomes, exomes and then the number of the common repeats between them.



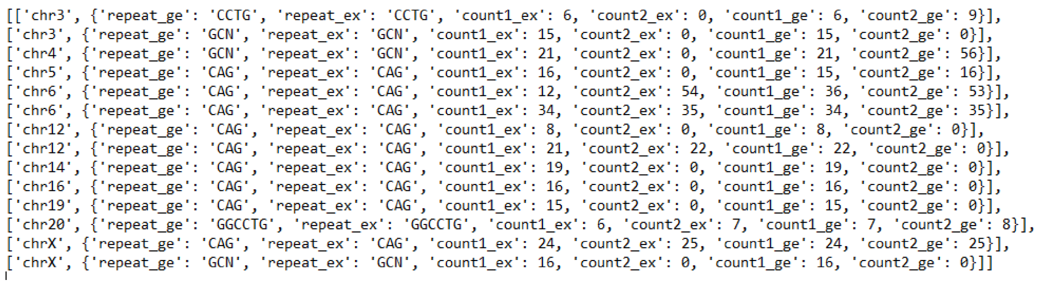
```
[['chr3', {'repeat_ge': 'GCN', 'repeat_ex': 'GCN', 'count1_ex': 15, 'count2_ex': 0, 'count1_ge': 15, 'count2_ge': 0}]
['chr4', {'repeat_ge': 'CAG', 'repeat_ex': 'CAG', 'count1_ex': 15, 'count2_ex': 0, 'count1_ge': 15, 'count2_ge': 0}]
['chr4', {'repeat_ge': 'GCN', 'repeat_ex': 'GCN', 'count1_ex': 21, 'count2_ex': 0, 'count1_ge': 21, 'count2_ge': 62}]
['chr5', {'repeat_ge': 'CAG', 'repeat_ex': 'CAG', 'count1_ex': 13, 'count2_ex': 14, 'count1_ge': 14, 'count2_ge': 16}]
['chr6', {'repeat_ge': 'CAG', 'repeat_ex': 'CAG', 'count1_ex': 14, 'count2_ex': 46, 'count1_ge': 14, 'count2_ge': 48}]
['chr6', {'repeat_ge': 'CAG', 'repeat_ex': 'CAG', 'count1_ex': 34, 'count2_ex': 0, 'count1_ge': 34, 'count2_ge': 0}]
['chr12', {'repeat_ge': 'CAG', 'repeat_ex': 'CAG', 'count1_ex': 22, 'count2_ex': 0, 'count1_ge': 22, 'count2_ge': 0}]
['chr14', {'repeat_ge': 'CAG', 'repeat_ex': 'CAG', 'count1_ex': 19, 'count2_ex': 24, 'count1_ge': 19, 'count2_ge': 24}]
['chr16', {'repeat_ge': 'CAG', 'repeat_ex': 'CAG', 'count1_ex': 16, 'count2_ex': 0, 'count1_ge': 16, 'count2_ge': 0}]
['chr19', {'repeat_ge': 'CAG', 'repeat_ex': 'CAG', 'count1_ex': 12, 'count2_ex': 0, 'count1_ge': 12, 'count2_ge': 0}]
['chr20', {'repeat_ge': 'GGCCTG', 'repeat_ex': 'GGCCTG', 'count1_ex': 3, 'count2_ex': 7, 'count1_ge': 3, 'count2_ge': 7}]
['chrX', {'repeat_ge': 'CAG', 'repeat_ex': 'CAG', 'count1_ex': 20, 'count2_ex': 21, 'count1_ge': 20, 'count2_ge': 21}]
['chrX', {'repeat_ge': 'GCN', 'repeat_ex': 'GCN', 'count1_ex': 16, 'count2_ex': 0, 'count1_ge': 16, 'count2_ge': 25}]]
```

(a) *Sample: Son*



```
[['chr3', {'repeat_ge': 'GCN', 'repeat_ex': 'GCN', 'count1_ex': 15, 'count2_ex': 0, 'count1_ge': 15, 'count2_ge': 27}]
['chr4', {'repeat_ge': 'CAG', 'repeat_ex': 'CAG', 'count1_ex': 15, 'count2_ex': 17, 'count1_ge': 15, 'count2_ge': 17}]
['chr4', {'repeat_ge': 'GCN', 'repeat_ex': 'GCN', 'count1_ex': 21, 'count2_ex': 0, 'count1_ge': 21, 'count2_ge': 49}]
['chr5', {'repeat_ge': 'CAG', 'repeat_ex': 'CAG', 'count1_ex': 11, 'count2_ex': 14, 'count1_ge': 11, 'count2_ge': 14}]
['chr6', {'repeat_ge': 'CAG', 'repeat_ex': 'CAG', 'count1_ex': 14, 'count2_ex': 30, 'count1_ge': 37, 'count2_ge': 55}]
['chr6', {'repeat_ge': 'CAG', 'repeat_ex': 'CAG', 'count1_ex': 34, 'count2_ex': 37, 'count1_ge': 34, 'count2_ge': 37}]
['chr12', {'repeat_ge': 'CAG', 'repeat_ex': 'CAG', 'count1_ex': 10, 'count2_ex': 0, 'count1_ge': 10, 'count2_ge': 0}]
['chr12', {'repeat_ge': 'CAG', 'repeat_ex': 'CAG', 'count1_ex': 21, 'count2_ex': 22, 'count1_ge': 21, 'count2_ge': 22}]
['chr14', {'repeat_ge': 'CAG', 'repeat_ex': 'CAG', 'count1_ex': 19, 'count2_ex': 24, 'count1_ge': 19, 'count2_ge': 24}]
['chr16', {'repeat_ge': 'CAG', 'repeat_ex': 'CAG', 'count1_ex': 15, 'count2_ex': 16, 'count1_ge': 15, 'count2_ge': 16}]
['chr19', {'repeat_ge': 'CAG', 'repeat_ex': 'CAG', 'count1_ex': 11, 'count2_ex': 0, 'count1_ge': 11, 'count2_ge': 0}]
['chr20', {'repeat_ge': 'GGCCTG', 'repeat_ex': 'GGCCTG', 'count1_ex': 3, 'count2_ex': 7, 'count1_ge': 3, 'count2_ge': 6}]
['chrX', {'repeat_ge': 'CAG', 'repeat_ex': 'CAG', 'count1_ex': 21, 'count2_ex': 29, 'count1_ge': 21, 'count2_ge': 29}]
['chrX', {'repeat_ge': 'GCN', 'repeat_ex': 'GCN', 'count1_ex': 16, 'count2_ex': 0, 'count1_ge': 16, 'count2_ge': 23}]
['chrX', {'repeat_ge': 'CGG', 'repeat_ex': 'CGG', 'count1_ex': 17, 'count2_ex': 0, 'count1_ge': 6, 'count2_ge': 0}]]
```

(b) *Sample: Mother*



```
[['chr3', {'repeat_ge': 'CCTG', 'repeat_ex': 'CCTG', 'count1_ex': 6, 'count2_ex': 0, 'count1_ge': 6, 'count2_ge': 9}],
['chr3', {'repeat_ge': 'GCN', 'repeat_ex': 'GCN', 'count1_ex': 15, 'count2_ex': 0, 'count1_ge': 15, 'count2_ge': 0}],
['chr4', {'repeat_ge': 'GCN', 'repeat_ex': 'GCN', 'count1_ex': 21, 'count2_ex': 0, 'count1_ge': 21, 'count2_ge': 56}],
['chr5', {'repeat_ge': 'CAG', 'repeat_ex': 'CAG', 'count1_ex': 16, 'count2_ex': 0, 'count1_ge': 15, 'count2_ge': 16}],
['chr6', {'repeat_ge': 'CAG', 'repeat_ex': 'CAG', 'count1_ex': 12, 'count2_ex': 54, 'count1_ge': 36, 'count2_ge': 53}],
['chr6', {'repeat_ge': 'CAG', 'repeat_ex': 'CAG', 'count1_ex': 34, 'count2_ex': 35, 'count1_ge': 34, 'count2_ge': 35}],
['chr12', {'repeat_ge': 'CAG', 'repeat_ex': 'CAG', 'count1_ex': 8, 'count2_ex': 0, 'count1_ge': 8, 'count2_ge': 0}],
['chr12', {'repeat_ge': 'CAG', 'repeat_ex': 'CAG', 'count1_ex': 21, 'count2_ex': 22, 'count1_ge': 22, 'count2_ge': 0}],
['chr14', {'repeat_ge': 'CAG', 'repeat_ex': 'CAG', 'count1_ex': 19, 'count2_ex': 0, 'count1_ge': 19, 'count2_ge': 0}],
['chr16', {'repeat_ge': 'CAG', 'repeat_ex': 'CAG', 'count1_ex': 16, 'count2_ex': 0, 'count1_ge': 16, 'count2_ge': 0}],
['chr19', {'repeat_ge': 'CAG', 'repeat_ex': 'CAG', 'count1_ex': 15, 'count2_ex': 0, 'count1_ge': 15, 'count2_ge': 0}],
['chr20', {'repeat_ge': 'GGCCTG', 'repeat_ex': 'GGCCTG', 'count1_ex': 6, 'count2_ex': 7, 'count1_ge': 7, 'count2_ge': 8}],
['chrX', {'repeat_ge': 'CAG', 'repeat_ex': 'CAG', 'count1_ex': 24, 'count2_ex': 25, 'count1_ge': 24, 'count2_ge': 25}],
['chrX', {'repeat_ge': 'GCN', 'repeat_ex': 'GCN', 'count1_ex': 16, 'count2_ex': 0, 'count1_ge': 16, 'count2_ge': 0}]]
```

(c) *Sample: Father*

Figure 5.4: Common RE between exome and genome made by EH tool.

The numbers reported in table 5.1, of Repeat Expansions in genome sequences, in exome sequences and the numbers of common detected repeats are computed as in the case of Expansion Hunter.

The generated files are shown below.

Looking at the file showing in figure 5.5, the common values and the number of the REs in the genome/exome samples, it can be seen that the number of repeated expansions it detects are nearly identical between genome and exome. Comparing these results with those obtained with Expansion Hunter it can be seen that the ability of Tredparse to identify REs in exomic sequences is greater. Thanks also to the use of a larger database that considers all the possible triplets within the regions defined previously for the diseases of interest.

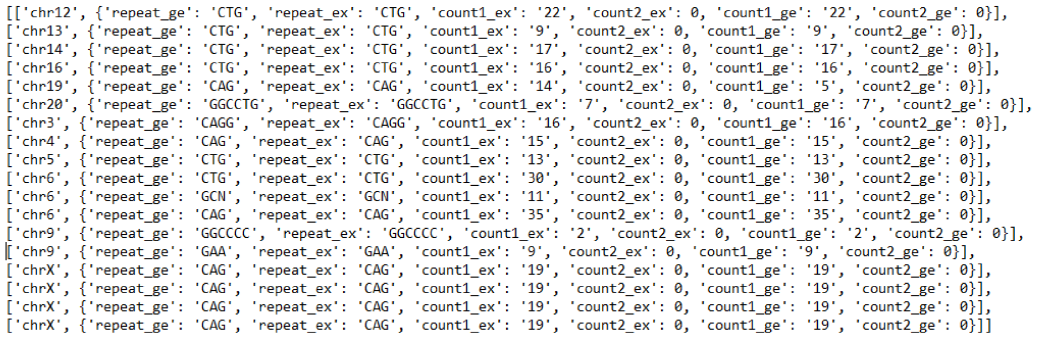
STRetch

STRetch already works with both exome and genome. In this case, the analysis is a bit different because this tool works with two different pipelines, one for exome sequences (WES) and one for genome sequences (WGS). We can note that at the code and input level, there are no substantial differences between the two types of pipeline. Therefore, it was decided to investigate the effective effectiveness in the use of a single pipeline in the face of a high number of common REs detected in the genome and exome of the same sample. Also, in order to optimize the time complexity of the analysis.

SAMPLE	SCRIPT GEN	SCRIPT EXO	GENOME	EXOME	COMMON
Proband	WES	WES	221	18	2
Proband	WGS	WES	84	18	17
Proband	WES	WGS	221	81	15
Proband	WGS	WGS	84	81	75

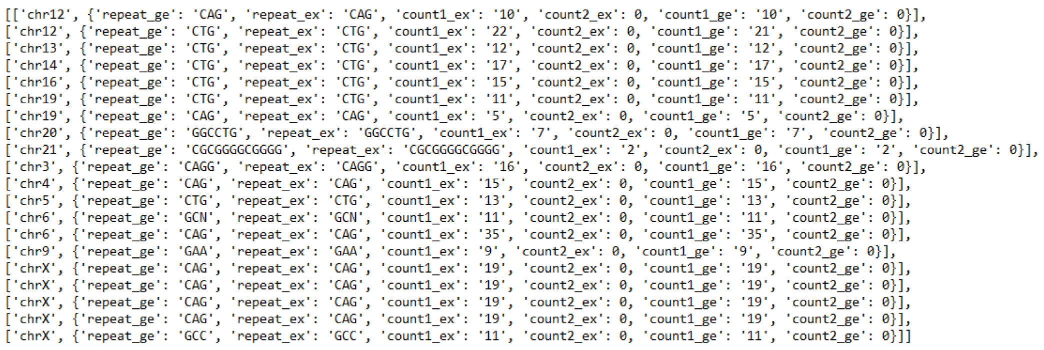
Table 5.3: For each test, it reports the pipeline used for genome and exome analysis, the number of repeats for genome and exome samples and the number of the common repeats between them (region and motif).

First of all, some preliminary remarks. Looking at the individual output files for each pipeline, it can be seen that the other number of repetitions detected by the WES pipeline in the genome is an error. In fact, many of these repetitions are composed of only one base. So it can already be concluded that using the WES pipeline on genomic sequences is a mistake.



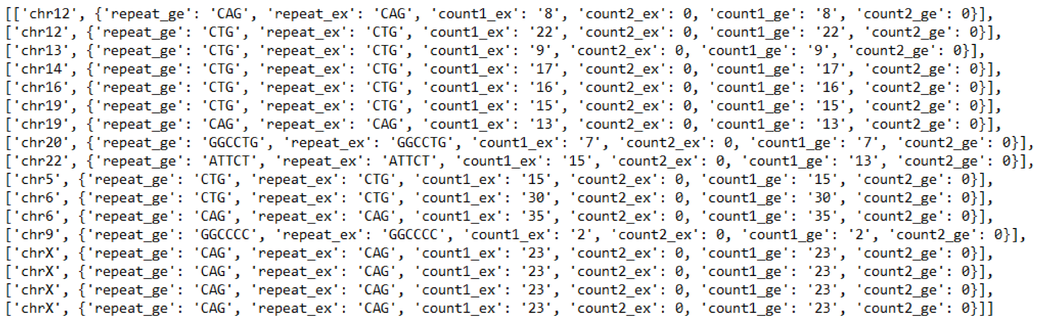
```
[['chr12', {'repeat_ge': 'CTG', 'repeat_ex': 'CTG', 'count1_ex': '22', 'count2_ex': 0, 'count1_ge': '22', 'count2_ge': 0}],
['chr13', {'repeat_ge': 'CTG', 'repeat_ex': 'CTG', 'count1_ex': '9', 'count2_ex': 0, 'count1_ge': '9', 'count2_ge': 0}],
['chr14', {'repeat_ge': 'CTG', 'repeat_ex': 'CTG', 'count1_ex': '17', 'count2_ex': 0, 'count1_ge': '17', 'count2_ge': 0}],
['chr16', {'repeat_ge': 'CTG', 'repeat_ex': 'CTG', 'count1_ex': '16', 'count2_ex': 0, 'count1_ge': '16', 'count2_ge': 0}],
['chr19', {'repeat_ge': 'CAG', 'repeat_ex': 'CAG', 'count1_ex': '14', 'count2_ex': 0, 'count1_ge': '5', 'count2_ge': 0}],
['chr20', {'repeat_ge': 'GGCCTG', 'repeat_ex': 'GGCCTG', 'count1_ex': '7', 'count2_ex': 0, 'count1_ge': '7', 'count2_ge': 0}],
['chr3', {'repeat_ge': 'CAGG', 'repeat_ex': 'CAGG', 'count1_ex': '16', 'count2_ex': 0, 'count1_ge': '16', 'count2_ge': 0}],
['chr4', {'repeat_ge': 'CAG', 'repeat_ex': 'CAG', 'count1_ex': '15', 'count2_ex': 0, 'count1_ge': '15', 'count2_ge': 0}],
['chr5', {'repeat_ge': 'CTG', 'repeat_ex': 'CTG', 'count1_ex': '13', 'count2_ex': 0, 'count1_ge': '13', 'count2_ge': 0}],
['chr6', {'repeat_ge': 'CTG', 'repeat_ex': 'CTG', 'count1_ex': '30', 'count2_ex': 0, 'count1_ge': '30', 'count2_ge': 0}],
['chr6', {'repeat_ge': 'GCN', 'repeat_ex': 'GCN', 'count1_ex': '11', 'count2_ex': 0, 'count1_ge': '11', 'count2_ge': 0}],
['chr6', {'repeat_ge': 'CAG', 'repeat_ex': 'CAG', 'count1_ex': '35', 'count2_ex': 0, 'count1_ge': '35', 'count2_ge': 0}],
['chr9', {'repeat_ge': 'GGCCCC', 'repeat_ex': 'GGCCCC', 'count1_ex': '2', 'count2_ex': 0, 'count1_ge': '2', 'count2_ge': 0}],
['chr9', {'repeat_ge': 'GAA', 'repeat_ex': 'GAA', 'count1_ex': '9', 'count2_ex': 0, 'count1_ge': '9', 'count2_ge': 0}],
['chrX', {'repeat_ge': 'CAG', 'repeat_ex': 'CAG', 'count1_ex': '19', 'count2_ex': 0, 'count1_ge': '19', 'count2_ge': 0}],
['chrX', {'repeat_ge': 'CAG', 'repeat_ex': 'CAG', 'count1_ex': '19', 'count2_ex': 0, 'count1_ge': '19', 'count2_ge': 0}],
['chrX', {'repeat_ge': 'CAG', 'repeat_ex': 'CAG', 'count1_ex': '19', 'count2_ex': 0, 'count1_ge': '19', 'count2_ge': 0}],
['chrX', {'repeat_ge': 'CAG', 'repeat_ex': 'CAG', 'count1_ex': '19', 'count2_ex': 0, 'count1_ge': '19', 'count2_ge': 0}]]
```

(a) *Sample: Proband*



```
[['chr12', {'repeat_ge': 'CAG', 'repeat_ex': 'CAG', 'count1_ex': '10', 'count2_ex': 0, 'count1_ge': '10', 'count2_ge': 0}],
['chr12', {'repeat_ge': 'CTG', 'repeat_ex': 'CTG', 'count1_ex': '22', 'count2_ex': 0, 'count1_ge': '21', 'count2_ge': 0}],
['chr13', {'repeat_ge': 'CTG', 'repeat_ex': 'CTG', 'count1_ex': '12', 'count2_ex': 0, 'count1_ge': '12', 'count2_ge': 0}],
['chr14', {'repeat_ge': 'CTG', 'repeat_ex': 'CTG', 'count1_ex': '17', 'count2_ex': 0, 'count1_ge': '17', 'count2_ge': 0}],
['chr16', {'repeat_ge': 'CTG', 'repeat_ex': 'CTG', 'count1_ex': '15', 'count2_ex': 0, 'count1_ge': '15', 'count2_ge': 0}],
['chr19', {'repeat_ge': 'CTG', 'repeat_ex': 'CTG', 'count1_ex': '11', 'count2_ex': 0, 'count1_ge': '11', 'count2_ge': 0}],
['chr19', {'repeat_ge': 'CAG', 'repeat_ex': 'CAG', 'count1_ex': '5', 'count2_ex': 0, 'count1_ge': '5', 'count2_ge': 0}],
['chr20', {'repeat_ge': 'GGCCTG', 'repeat_ex': 'GGCCTG', 'count1_ex': '7', 'count2_ex': 0, 'count1_ge': '7', 'count2_ge': 0}],
['chr21', {'repeat_ge': 'CGCGGGGCGGGG', 'repeat_ex': 'CGCGGGGCGGGG', 'count1_ex': '2', 'count2_ex': 0, 'count1_ge': '2', 'count2_ge': 0}],
['chr3', {'repeat_ge': 'CAGG', 'repeat_ex': 'CAGG', 'count1_ex': '16', 'count2_ex': 0, 'count1_ge': '16', 'count2_ge': 0}],
['chr4', {'repeat_ge': 'CAG', 'repeat_ex': 'CAG', 'count1_ex': '15', 'count2_ex': 0, 'count1_ge': '15', 'count2_ge': 0}],
['chr5', {'repeat_ge': 'CTG', 'repeat_ex': 'CTG', 'count1_ex': '13', 'count2_ex': 0, 'count1_ge': '13', 'count2_ge': 0}],
['chr6', {'repeat_ge': 'GCN', 'repeat_ex': 'GCN', 'count1_ex': '11', 'count2_ex': 0, 'count1_ge': '11', 'count2_ge': 0}],
['chr6', {'repeat_ge': 'CAG', 'repeat_ex': 'CAG', 'count1_ex': '35', 'count2_ex': 0, 'count1_ge': '35', 'count2_ge': 0}],
['chr9', {'repeat_ge': 'GAA', 'repeat_ex': 'GAA', 'count1_ex': '9', 'count2_ex': 0, 'count1_ge': '9', 'count2_ge': 0}],
['chrX', {'repeat_ge': 'CAG', 'repeat_ex': 'CAG', 'count1_ex': '19', 'count2_ex': 0, 'count1_ge': '19', 'count2_ge': 0}],
['chrX', {'repeat_ge': 'CAG', 'repeat_ex': 'CAG', 'count1_ex': '19', 'count2_ex': 0, 'count1_ge': '19', 'count2_ge': 0}],
['chrX', {'repeat_ge': 'CAG', 'repeat_ex': 'CAG', 'count1_ex': '19', 'count2_ex': 0, 'count1_ge': '19', 'count2_ge': 0}],
['chrX', {'repeat_ge': 'CAG', 'repeat_ex': 'CAG', 'count1_ex': '19', 'count2_ex': 0, 'count1_ge': '19', 'count2_ge': 0}],
['chrX', {'repeat_ge': 'GCC', 'repeat_ex': 'GCC', 'count1_ex': '11', 'count2_ex': 0, 'count1_ge': '11', 'count2_ge': 0}]]
```

(b) *Sample: Mother*



```
[['chr12', {'repeat_ge': 'CAG', 'repeat_ex': 'CAG', 'count1_ex': '8', 'count2_ex': 0, 'count1_ge': '8', 'count2_ge': 0}],
['chr12', {'repeat_ge': 'CTG', 'repeat_ex': 'CTG', 'count1_ex': '22', 'count2_ex': 0, 'count1_ge': '22', 'count2_ge': 0}],
['chr13', {'repeat_ge': 'CTG', 'repeat_ex': 'CTG', 'count1_ex': '9', 'count2_ex': 0, 'count1_ge': '9', 'count2_ge': 0}],
['chr14', {'repeat_ge': 'CTG', 'repeat_ex': 'CTG', 'count1_ex': '17', 'count2_ex': 0, 'count1_ge': '17', 'count2_ge': 0}],
['chr16', {'repeat_ge': 'CTG', 'repeat_ex': 'CTG', 'count1_ex': '16', 'count2_ex': 0, 'count1_ge': '16', 'count2_ge': 0}],
['chr19', {'repeat_ge': 'CTG', 'repeat_ex': 'CTG', 'count1_ex': '15', 'count2_ex': 0, 'count1_ge': '15', 'count2_ge': 0}],
['chr19', {'repeat_ge': 'CAG', 'repeat_ex': 'CAG', 'count1_ex': '13', 'count2_ex': 0, 'count1_ge': '13', 'count2_ge': 0}],
['chr20', {'repeat_ge': 'GGCCTG', 'repeat_ex': 'GGCCTG', 'count1_ex': '7', 'count2_ex': 0, 'count1_ge': '7', 'count2_ge': 0}],
['chr22', {'repeat_ge': 'ATTCT', 'repeat_ex': 'ATTCT', 'count1_ex': '15', 'count2_ex': 0, 'count1_ge': '13', 'count2_ge': 0}],
['chr5', {'repeat_ge': 'CTG', 'repeat_ex': 'CTG', 'count1_ex': '15', 'count2_ex': 0, 'count1_ge': '15', 'count2_ge': 0}],
['chr6', {'repeat_ge': 'CTG', 'repeat_ex': 'CTG', 'count1_ex': '30', 'count2_ex': 0, 'count1_ge': '30', 'count2_ge': 0}],
['chr6', {'repeat_ge': 'CAG', 'repeat_ex': 'CAG', 'count1_ex': '35', 'count2_ex': 0, 'count1_ge': '35', 'count2_ge': 0}],
['chr9', {'repeat_ge': 'GGCCCC', 'repeat_ex': 'GGCCCC', 'count1_ex': '2', 'count2_ex': 0, 'count1_ge': '2', 'count2_ge': 0}],
['chrX', {'repeat_ge': 'CAG', 'repeat_ex': 'CAG', 'count1_ex': '23', 'count2_ex': 0, 'count1_ge': '23', 'count2_ge': 0}],
['chrX', {'repeat_ge': 'CAG', 'repeat_ex': 'CAG', 'count1_ex': '23', 'count2_ex': 0, 'count1_ge': '23', 'count2_ge': 0}],
['chrX', {'repeat_ge': 'CAG', 'repeat_ex': 'CAG', 'count1_ex': '23', 'count2_ex': 0, 'count1_ge': '23', 'count2_ge': 0}],
['chrX', {'repeat_ge': 'CAG', 'repeat_ex': 'CAG', 'count1_ex': '23', 'count2_ex': 0, 'count1_ge': '23', 'count2_ge': 0}]]
```

(c) *Sample: Father*

Figure 5.5: Common RE between exome and genome made by Tredparse tool.

Furthermore, it should be noted that tests have been done on all chromosomes and not only in the defined regions of interest in this project. This is to have a more complete comparison. Also, we should remember that STretch checks for any type of repetition in the sequence to be analyzed.

Given the results obtained in the third case, that is what is indicated in the documentation as a standard procedure, it can be noted that the REs identified in the genome are numerically much greater than those identified in the exome. However, the percentage of exomic REs also present in the genome is almost 95%. Now, let's compare the other interesting case with this one, the last one, where the WGS pipeline was used for both types of sequences.

In this case, the percentage of common repetitions is almost equal to that of the previous case (93%). So even at the computational level, it is possible to use the WGS pipeline also on exoma. In addition to having a high number of repetitions in common, looking at the extracted files by superimposing the outputs of the genome and exome, it can be seen that the detected values are very similar if not the same.

The only differences concern for the intronic regions, obviously not considered in the exomic analyzes. The genome can be analyzed for greater completeness and for the detection of repeat expansions in intronic regions. Regarding STRetch, it was decided to use the genomic pipeline for both genome and exome processing. However, this guarantees a good result with reasonable computation times. In figure 5.5 are reported the relative files.

At the end, it can be concluded that the analysis can also be done on genomic data, but with some implementation and theoretical shrewdness.

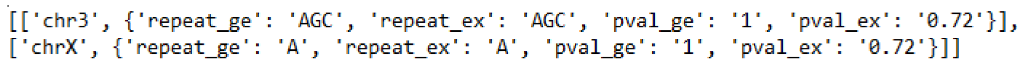
5.2.2 Whole pipeline tests

When the pipeline was implemented entirely, its correctness and performance are tested on real positive samples both exome and genome.

The samples used to verify the whole pipeline are reported below. The detected REs are pathological for a specific disease if they are in the corresponding region and the number of repeats falls within the range of not normality. The samples below, being positive, possess specific REs also reported on the list.

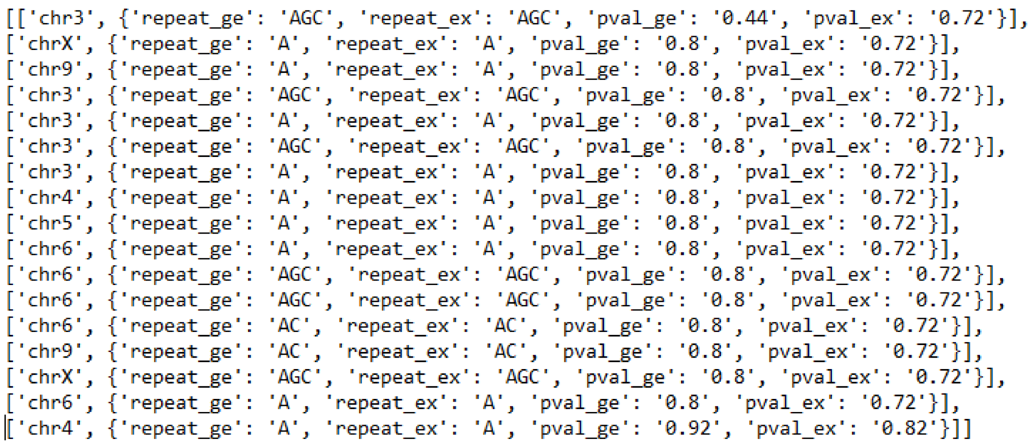
The samples are:

- Exome sequence positive to Huntigton's disease. Pathological RE in the HTT gene (chromosome 4) with number of repeats greater than 35. This diseases is dominant and exonic.



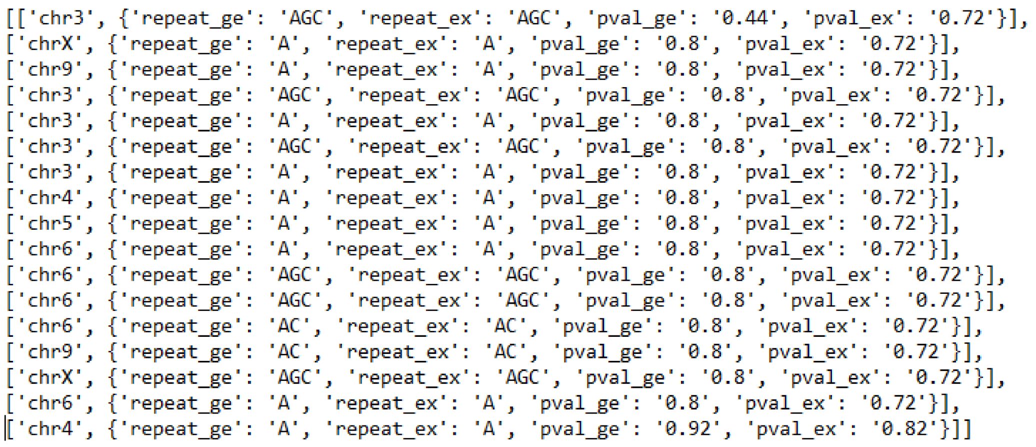
```
[['chr3', {'repeat_ge': 'AGC', 'repeat_ex': 'AGC', 'pval_ge': '1', 'pval_ex': '0.72'}],
['chrX', {'repeat_ge': 'A', 'repeat_ex': 'A', 'pval_ge': '1', 'pval_ex': '0.72'}]]
```

(a) *Sample: Proband, wes-wes*



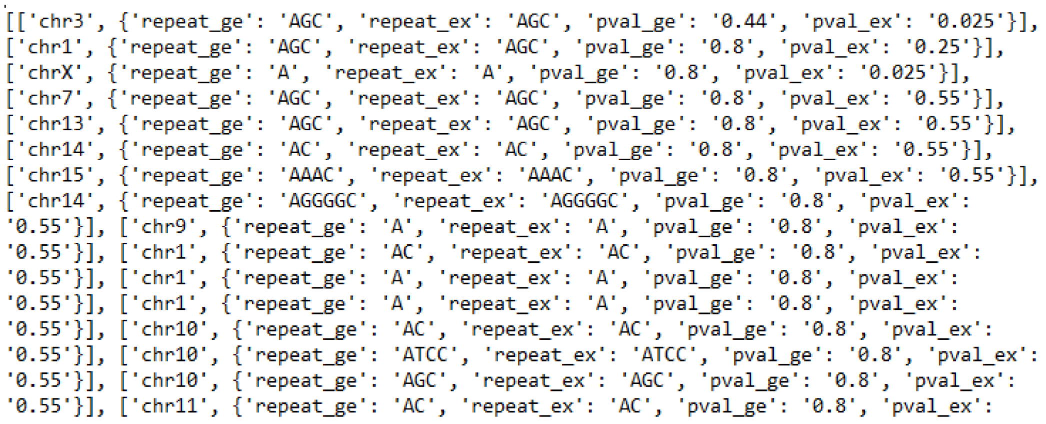
```
[['chr3', {'repeat_ge': 'AGC', 'repeat_ex': 'AGC', 'pval_ge': '0.44', 'pval_ex': '0.72'}],
['chrX', {'repeat_ge': 'A', 'repeat_ex': 'A', 'pval_ge': '0.8', 'pval_ex': '0.72'}],
['chr9', {'repeat_ge': 'A', 'repeat_ex': 'A', 'pval_ge': '0.8', 'pval_ex': '0.72'}],
['chr3', {'repeat_ge': 'AGC', 'repeat_ex': 'AGC', 'pval_ge': '0.8', 'pval_ex': '0.72'}],
['chr3', {'repeat_ge': 'A', 'repeat_ex': 'A', 'pval_ge': '0.8', 'pval_ex': '0.72'}],
['chr3', {'repeat_ge': 'AGC', 'repeat_ex': 'AGC', 'pval_ge': '0.8', 'pval_ex': '0.72'}],
['chr3', {'repeat_ge': 'A', 'repeat_ex': 'A', 'pval_ge': '0.8', 'pval_ex': '0.72'}],
['chr4', {'repeat_ge': 'A', 'repeat_ex': 'A', 'pval_ge': '0.8', 'pval_ex': '0.72'}],
['chr5', {'repeat_ge': 'A', 'repeat_ex': 'A', 'pval_ge': '0.8', 'pval_ex': '0.72'}],
['chr6', {'repeat_ge': 'A', 'repeat_ex': 'A', 'pval_ge': '0.8', 'pval_ex': '0.72'}],
['chr6', {'repeat_ge': 'AGC', 'repeat_ex': 'AGC', 'pval_ge': '0.8', 'pval_ex': '0.72'}],
['chr6', {'repeat_ge': 'AGC', 'repeat_ex': 'AGC', 'pval_ge': '0.8', 'pval_ex': '0.72'}],
['chr6', {'repeat_ge': 'AC', 'repeat_ex': 'AC', 'pval_ge': '0.8', 'pval_ex': '0.72'}],
['chr9', {'repeat_ge': 'AC', 'repeat_ex': 'AC', 'pval_ge': '0.8', 'pval_ex': '0.72'}],
['chrX', {'repeat_ge': 'AGC', 'repeat_ex': 'AGC', 'pval_ge': '0.8', 'pval_ex': '0.72'}],
['chr6', {'repeat_ge': 'A', 'repeat_ex': 'A', 'pval_ge': '0.8', 'pval_ex': '0.72'}],
['chr4', {'repeat_ge': 'A', 'repeat_ex': 'A', 'pval_ge': '0.92', 'pval_ex': '0.82'}]]
```

(b) *Sample: Proband, wgs-wes*



```
[['chr3', {'repeat_ge': 'AGC', 'repeat_ex': 'AGC', 'pval_ge': '0.44', 'pval_ex': '0.72'}],
['chrX', {'repeat_ge': 'A', 'repeat_ex': 'A', 'pval_ge': '0.8', 'pval_ex': '0.72'}],
['chr9', {'repeat_ge': 'A', 'repeat_ex': 'A', 'pval_ge': '0.8', 'pval_ex': '0.72'}],
['chr3', {'repeat_ge': 'AGC', 'repeat_ex': 'AGC', 'pval_ge': '0.8', 'pval_ex': '0.72'}],
['chr3', {'repeat_ge': 'A', 'repeat_ex': 'A', 'pval_ge': '0.8', 'pval_ex': '0.72'}],
['chr3', {'repeat_ge': 'AGC', 'repeat_ex': 'AGC', 'pval_ge': '0.8', 'pval_ex': '0.72'}],
['chr3', {'repeat_ge': 'A', 'repeat_ex': 'A', 'pval_ge': '0.8', 'pval_ex': '0.72'}],
['chr4', {'repeat_ge': 'A', 'repeat_ex': 'A', 'pval_ge': '0.8', 'pval_ex': '0.72'}],
['chr5', {'repeat_ge': 'A', 'repeat_ex': 'A', 'pval_ge': '0.8', 'pval_ex': '0.72'}],
['chr6', {'repeat_ge': 'A', 'repeat_ex': 'A', 'pval_ge': '0.8', 'pval_ex': '0.72'}],
['chr6', {'repeat_ge': 'AGC', 'repeat_ex': 'AGC', 'pval_ge': '0.8', 'pval_ex': '0.72'}],
['chr6', {'repeat_ge': 'AGC', 'repeat_ex': 'AGC', 'pval_ge': '0.8', 'pval_ex': '0.72'}],
['chr6', {'repeat_ge': 'AC', 'repeat_ex': 'AC', 'pval_ge': '0.8', 'pval_ex': '0.72'}],
['chr9', {'repeat_ge': 'AC', 'repeat_ex': 'AC', 'pval_ge': '0.8', 'pval_ex': '0.72'}],
['chrX', {'repeat_ge': 'AGC', 'repeat_ex': 'AGC', 'pval_ge': '0.8', 'pval_ex': '0.72'}],
['chr6', {'repeat_ge': 'A', 'repeat_ex': 'A', 'pval_ge': '0.8', 'pval_ex': '0.72'}],
['chr4', {'repeat_ge': 'A', 'repeat_ex': 'A', 'pval_ge': '0.92', 'pval_ex': '0.82'}]]
```

(c) *Sample: Proband, wes-wgs*



```
[['chr3', {'repeat_ge': 'AGC', 'repeat_ex': 'AGC', 'pval_ge': '0.44', 'pval_ex': '0.025'}],
['chr1', {'repeat_ge': 'AGC', 'repeat_ex': 'AGC', 'pval_ge': '0.8', 'pval_ex': '0.25'}],
['chrX', {'repeat_ge': 'A', 'repeat_ex': 'A', 'pval_ge': '0.8', 'pval_ex': '0.025'}],
['chr7', {'repeat_ge': 'AGC', 'repeat_ex': 'AGC', 'pval_ge': '0.8', 'pval_ex': '0.55'}],
['chr13', {'repeat_ge': 'AGC', 'repeat_ex': 'AGC', 'pval_ge': '0.8', 'pval_ex': '0.55'}],
['chr14', {'repeat_ge': 'AC', 'repeat_ex': 'AC', 'pval_ge': '0.8', 'pval_ex': '0.55'}],
['chr15', {'repeat_ge': 'AAAC', 'repeat_ex': 'AAAC', 'pval_ge': '0.8', 'pval_ex': '0.55'}],
['chr14', {'repeat_ge': 'AGGGGC', 'repeat_ex': 'AGGGGC', 'pval_ge': '0.8', 'pval_ex':
'0.55'}], ['chr9', {'repeat_ge': 'A', 'repeat_ex': 'A', 'pval_ge': '0.8', 'pval_ex':
'0.55'}], ['chr1', {'repeat_ge': 'AC', 'repeat_ex': 'AC', 'pval_ge': '0.8', 'pval_ex':
'0.55'}], ['chr1', {'repeat_ge': 'A', 'repeat_ex': 'A', 'pval_ge': '0.8', 'pval_ex':
'0.55'}], ['chr1', {'repeat_ge': 'A', 'repeat_ex': 'A', 'pval_ge': '0.8', 'pval_ex':
'0.55'}], ['chr10', {'repeat_ge': 'AC', 'repeat_ex': 'AC', 'pval_ge': '0.8', 'pval_ex':
'0.55'}], ['chr10', {'repeat_ge': 'ATCC', 'repeat_ex': 'ATCC', 'pval_ge': '0.8', 'pval_ex':
'0.55'}], ['chr10', {'repeat_ge': 'AGC', 'repeat_ex': 'AGC', 'pval_ge': '0.8', 'pval_ex':
'0.55'}], ['chr11', {'repeat_ge': 'AC', 'repeat_ex': 'AC', 'pval_ge': '0.8', 'pval_ex':
```

(d) *Sample: Proband, wgs-wgs*

Figure 5.6: Common RE between exome and genome made by STRetch pipelines.

- Exome and genome sequences positive to Spinocerebellar Ataxia of type 1 (SCA1). Both samples have pathological RE in the ATX1 gene (chromosome 6) with number of repeats between 44 and 83. This diseases is dominant and exonic.

- Exome and genome sequences positive to Spinocerebellar Ataxia of type 3 (SCA3). These sequences have pathological RE in the ATX3 gene (chromosome 14) with number of repeats between 54 and 86. This diseases is dominant and exonic.

- Exome and genome sequences positive to Spinocerebellar Ataxia of type 7 (SCA7). Both samples have pathological RE in the ATX7 gene (chromosome 3) with number of repeats greater than 33. This pathology is dominant and exonic.

The positivity of these samples is verified.

To test the accuracy of the developed method, it was run on each sample and then compared the output with known pathology (and premutation) results.

After the analysis of these samples using the implemented method, two separate files are reported in the output for each of them. The first file reports the list of REs detected as pathological after all the various comparisons made by the pipeline, while the second file reports the list of REs recognized as being in the premutation zone. This last file is important as a medical result for future generations of the individual whose sample is being examined.

Both output files report the same information about the REs in their list:

- Pathology: name of the pathology associated with the Repeat Expansion.

- Motif: repeated nucleotide sequence.

- Repeats, allele 1: number of the motif repetitions into the specific region in the first allele. This value is the mean of the number of repeats computed by Expansion Hunter and Tredparse, on the first allele.

- Repeats, allele 2: number of the nucleotide sequence repeats into the specific region in the second allele. The value is computed in the same way as above but w.t.r. the second allele.

- P value: This value is reported in output by STRetch program. Because of a different RE identification approach, this tool doesn't consider the two alleles separately, and the results reported are statistics. Therefore, this value was considered separately from the repeat counts and it's used as a measure of the goodness of the results about the reported RE with respect to the region in which it is found.

- Pathological/Premunition allele: For each RE identified, the method checks whether the associated disease is recessive or dominant. Then it checks whether at least one (dominant) or both (recessive) repeat values fall within the pathological or premutation ranges.

The results obtained on the exome and genome sequences of the above samples are shown below in tabular form. For each sample, the information, that are contained in the respective output files after their analysis by the developed method, is reported in the tables. The results are then accompanied with some remarks due to the pipeline reported different results between exome and genome sequences.

Exome

SAMPLE	DISEASE	MOTIF	ALLELE 1	ALLELE 2	PAT ALLELE
HD	HD	CAG	17	44	Allele 2
SCA1	SCA1	CAG	25.5	49	Allele 2
SCA3	SCA3	CAG	19	51.5	Allele 2
SCA7	SCA7	CAG	7	46.5	Allele 2

Table 5.4: Patological REs identified by the pipeline in the exome samples.

Looking at the results obtained, it can be seen that the pipeline correctly detected pathological REs in the correct regions and with the expected values, as shown in the table. In fact, the pathologies associated with these REs (second column) are the same as those known for each sample (first column).

It should be noted that since all these diseases are dominant, each sample is positive even if there is only one allele per sample that falls into the pathological range (last column). In detail, all samples are pathological with respect to the second allele of the gene whose mutation leads to the specific disease. In addition, the counts of each reported repetition are validated by the corresponding p value calculated in that region by the STRetch program.

Finally, concerning the REs that fall in the premutation zone, none was detected and the respective output files are empty. Therefore, no value was reported in the table.

Genome

For the genome sequences, the values reported for each allele are those calculated from the pipeline even if they are not pathological. These REs refer to the known diseases for each sample.

In these cases, the output files of the whole pipeline are both empty, so no REs were detected either pathological or in the premutation zone. In fact, for all the genome samples the pipeline didn't detect any REs diseases.

SAMPLE	MOTIF	ALLELE 1	ALLELE 2
SCA1	CAG	32.5	40
SCA3	CAG	19	49.5
SCA7	CAG	7	31.5

Table 5.5: Values of some specific REs identified by the pipeline in the genome samples.

This fact is due to the values found by the two tools on the possible pathological allele being quite different and the average is below the ranges of pathology and even premutation.

In fact, looking at the individual output files of the tools, before the comparison made by the pipeline, the following observation can be made. It is possible that the different tools used have different sensitivity, especially in the face of lower genome sample coverage. Indeed, Expansion Hunter results alone are quite correct and would detect pathological REs but Tredparse results are much lower.

The reason why, even though the tools and pipeline were designed to analyze both exome and genome sequences, maybe lies in the different degree of coverage.

It can be seen that with other samples in the dataset the results obtained on the two sample types are quite good, but in the case of positive sequences, correct REs are identified but on average with lower values in the genome than in the exome. Samples sequenced as genomes have significantly less coverage than the same samples sequenced as exomas. This means fewer reads mapping each position of the genomic sequence and less accuracy in the final aligned sequence. Thus, it can be assumed that the identification and evaluation of the number of repetitions of a given genomic sequence in a specific region are more difficult in genomic sequences.

In conclusion, even if the pipeline is developed in order to identify in the best way also in genomic sequences, in this case it has not led to satisfactory results.

Chapter 6

Conclusions

This project followed the findings reported in the publication [21] in which a collection of bioinformatics tools were used for the identification and analysis of Repeat Expansions present in the samples under investigation. The results of this study highlight the importance of using multiple tools to provide redundancy in the data analysis pipeline.

To conclude, we would like to focus on some important aspects that emerged during the development of this pipeline, identifying positive and critical aspects.

First of all, regarding the design and development of the method:

During realization, no particular problems were encountered at the design and development level. Many tests were carried out to test the tools and search for those most useful for the purpose of the pipeline. In addition, particular care was taken in the study and subsequent setting of the parameters so that the method could be used with both genome and exome sequences, as the programmes used were not set up.

One of the difficulties encountered relates to the use and integration of the results of the STRetch tool with the rest of the pipeline. In fact, there were many problems in the testing phase of STRetch at the executive level and in setting up the different pipelines used in the programme. Moreover, the output obtained is structurally different from those obtained by the other tools. The RE identified are approximately the same, but the counts are reported in statistical terms. The results were nevertheless taken into account because it provides a different view in the analysis of RE. After several tests, it was decided to mainly use the other two tools for the evaluation of Repeat Expansions.

It is important to highlight the need for a pre-processing phase of the BAM file with the genome or exome sequence of the sample. In this phase, the sorting and indexing of the file containing the aligned sequence are to be carried out. Moreover, after the performance verification step of the developed pipeline, it was noticed that it is necessary to have samples with high coverage in order to detect and evaluate the length of expansions more accurately.

Then, compared to the results obtained in tests with real samples:

- The samples used are related to individuals.
- Let us consider the part of the dataset with which parameters were set for both exome and genome sequences. It can be seen that there were no false positives compared to the negative samples used.
- Tests carried out on exome sequences positive for RE diseases have produced excellent outcomes reporting 100% of the expected results.
- Tests on genome sequences did not detect the expected pathologies of the samples analysed. However, it can be highlighted, by looking at the reported data, that the RE counts detected are very close to the pathological ranges. Future tests are possible, considering different coverage and improvements in the pipeline. It is worth noting that genome sequences with 40x coverage are already very high and expensive in terms of time and cost of analysis.
- From a performance point of view, it can be seen that, with sequences matching the requirements (coverage and sequencing type), the analysis is fast and on a large gene pool. In fact, as desired, the identification of REs using the developed method can be performed in approximately 10/12 hours (starting from raw BAM files) and considers many candidate genes in parallel.

In conclusion, the developed method is able to accurately identify Repeat Expansions, giving them pathological significance.

The degree of reliability on exomic sequences is very high, but future improvements and possible tests could be made on its use with genomic sequences. Indeed, by refining this part too, it is possible to analyse samples against intronic diseases in an efficient way. Possible trials on genomic samples involve different levels of coverage and types of sequencing. The parameters used are set to the optimum. Furthermore, it is possible to examine the possibility of other methods for merging the results obtained from the individual tools, since Expansion Hunter has a higher accuracy than Tredparse and STRetch.

Bibliography

[1] Andrea E Murmann, Jindan Yu, Puneet Opal, Marcus E Peter, ”Trinucleotide Repeat Expansion Diseases, RNAi, and Cancer”, Trends Cancer, 2018.

[2] Depienne Christel and Mandel Jean Louis, ”30 years of repeat expansion disorders: What have we learned and what are the remaining challenges?”, Institute of Human Genetics, 2021.

[3] Henry Paulson, ”Repeat expansion diseases”, Handb Clin Neurol, 2019.

[4] OMIM, Huntington’s disease, `https://www.omim.org/entry/143100`.

[5] Henry L. Paulson, ”The Spinocerebellar Ataxias”, J Neuroophthalmol,2010.

[6] OMIM, Fragile X syndrome, `https://www.omim.org/entry/300624`.

[7] Subbaya Subramanian, Rakesh K Mishra and Lalji Singh, ”Genome-wide analysis of microsatellite repeats in humans: their abundance and density in specific genomic regions”, BMC Bioinformatics, 2003.

[8] Qian Liu, ”Genome-wide detection of short tandem repeat expansions by long-read sequencing”, Genome Res, 2020.

[9] Dolzhenko Egor, ”Detection of long repeat expansions from PCR-free whole-genome sequence data”, J Hum Genet, 2017.

[10] Dolzhenko Egor, Github code about Expansion Hunter, 2021, `https://github.com/Illumina/ExpansionHunter`.

[11] Fearnley, ”Ultrafast, alignment-free detection of repeat expansions in NGS and RNAseq data”, 2021.

[12] Fearnley, Github code about SuperSTR, 2021, `https://github.com/bahlolab/superSTR`.

[13] M. Tankard Rick, ”Detecting Expansions of Tandem Repeats in Cohorts Sequenced with Short-Read Sequencing Data”, Genome Biol, 2018.

[14] M. Tankard, Rick, Github code about ExSTRa, 2021, `https://github.com/bahlolab/exSTRa`.

[15] Harriet Dashnow, Monkol Lek, ”STRetch: detecting and discovering pathogenic short tandem repeat expansions”, 2018.

[16] Tang Haibao, ”Detecting Expansions of Tandem Repeats in Cohorts Sequenced with Short-Read Sequencing Data”, Genome Med, 2018.

[17] Tang Haibao, Github code about Tredparse, 2019, `https://github.com/humanlongevity/tredparse`.

[18] European Genome phenome Archive, EGA dataset 00001003512, `https://ega-archive.org/datasets/EGAD00001003512`.

[19] European Genome phenome Archive, EGA dataset 00001003562, `https://ega-archive.org/datasets/EGAD00001003562`.

[20] Rajan-Babu Indhu-Shree, Peng Junran and Chiu Readman, ”Genome-Wide Sequencing as a First-Tier Screening Test for Short Tandem Repeat Expansions”, Genome Med, 2020.

[21] Haloom Rafehi, ”Bioinformatics-Based Identification of Expanded Repeats: A Non-reference Intronic Pentamer Expansion in RFC1 Causes CANVAS”, 2019.

List of Figures

List of Tables

Acknowledgements

First of all, I immensely thank my parents Lorena and Gianluigi who have always been close to me, supporting me when I needed and directing me towards the right choices. I also thank my brother Niccolò, who always put up with me even at times when I was most anxious.

A big thank to my friends who have made this path lighter with their joy and the sharing of anxieties and fears. Thanks for the comparisons, but also for the breaks in company and the jokes.

Thanks go to Professor Vandin, who with his professionalism advised and supported me during my thesis project. I would like to thank the company R&I Genetics that gave me the opportunity to do my internship and develop my thesis project in a positive and challenging environment. Finally, I thank all the friends and relatives who have helped, encouraged and made me smiling.