Genome sequencing data analysis: genetic causes of Parkinson's disease and hypertrophic cardiomyopathy

Adèle Mortier
adele.mortier@gmail.com

Abstract

Parkinson's disease (PD) is a long term disorder of the central nervous system that mainly affects the motor system[1]. The symptoms generally come on slowly over time. The cause of Parkinson's disease is generally unknown, but believed to involve both genetic and environmental factors.

Hypertrophic cardiomyopathy (HCM) is a primary disease of the myocardium (the muscle of the heart) in which a portion of the myocardium is hypertrophied (thickened) without any obvious cause, creating functional impairment of the cardiac muscle[2, 3, 4, 5, 6]. The occurrence of hypertrophic cardiomyopathy is a significant cause of sudden cardiac death. The disease can be inherited as an autosomal dominant trait, or come from a *de novo* mutation.

In this analysis, we will focus on the genetic factors that could possibly be a source of PD and HCM. To this aim, several steps have been performed, from preprocessing tasks on raw genomes to mutation analysis, including indexing and alignement.

Keywords:
genomics, Parkinson's disease, Hypertrophic cardiomyopathy, read-alignment, bioinformatics

1. Introduction

The study involved 30 patients from a local hospital in Novosibirsk, who developped Parkinson's disease or Hypertrophic cardiomyopathy. More precisely, 15 patients had PD and the 15 others had HCM. They had given some

blood samples that were sequenced in Moscow with Illumina HiSeq machines. Our goal was to compare the sequenced genomes of tese 30 patients with a reference genome (for us, the human genome called hg38). But this comparison is not self-evident, for roughly two reasons. The first was that the sequenced data send from Moscow were raw data, that means, data with low quality (uncertain) reads and unwanted sequences that did not really came from the patients genomes. The second was that read-alignement remains today a difficult problem, which leads to different trials with different softwares to determine the best way to map DNA sequences. Our study has been separated into three parts. Firstly, we preprocessed the data by filtering and trimmming, then we mapped the reads against the reference genome with several tools, and evaluated their accuracy. Finally, we tried to find variants, made stats on them and analyzed the results.

2. Preprocessing tasks

2.1. Quality check

The first step we performed was a global quality check of our data, to evaluate namely the quality distribution amongst the reads, and the unexpected presence of overrepresenteed k-mers or sequences. More precisely, our data consisted of 60 .fastq files, two for each patient. Indeed, the Illumina HiSeq technology allowed us to sequence pair-end read, which means that the machine was reading simultaneously both end of the portion of DNA sequence, one "forwards", the other "backwards". The format of our files was as follows:

```
<two letters><four digits>_<read direction>.fastq
```

where the read direction was encoded by a 1 if the read as "forwards" and by a 2 if the read was "backwards". We then run a routine to automatically check the quality of all reads, using the FastQC program:

Listing 1: FastQC analysis

```
#!/bin/bash
for f in *.fastq
do
        echo $f
        fastqc $f
done
```

The results obtained consisted of several graphics; we chose to outline a couple of them, which semmed to pose problems:

Figure 1: FastQC results for `as1017_1`

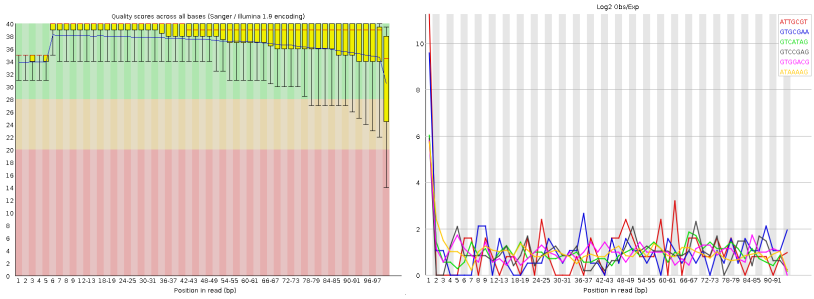
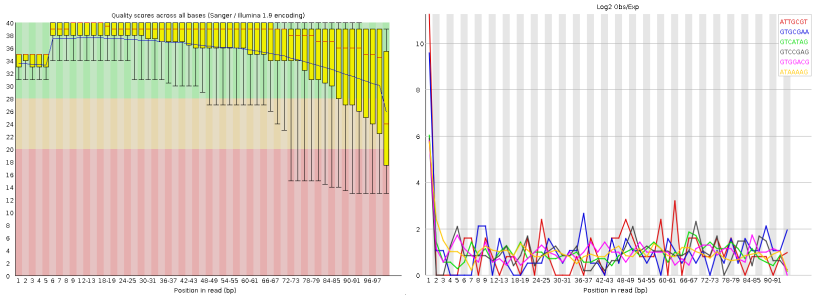


Figure 2: FastQC results for `as1017_2`



We noticed for example that the quality distributions were more likely to be bad at the end of each read That is why we decided to perform a quality filtering before mapping these seqeunces We also remarked, for a few samples, that the k-mer distribution could present some unexpected peaks amid the relative flat repartition of the graph. This kind of anomaly made us suppose (altough there was no really overrepreseted sequences in our reads) that an adapter clipping should be necessary during the next steps.

2.2. Quality filtering

We tried to reduce the average width of the quantiles we could see in our quality distribution graphics, in order to have the majority of our bases located in the green area, which corresponds to a good reliance in the base reading. To eliminate "bad reads", we firstly performed quality filtering, which removes sequences with low base quality scores, short reads, and those with features suggestive of higher error probability. Thus we used a quality fltering tool from the FASTX-Toolkit[7], called `fastq_quality_filter`. The command we run was:

```
fastq_quality_filter -q 20 -p 80 -i <input> -o <output> -Q33
```

Where the `-q` parameter corresponds to the minimum quality score to keep (here, `20`) and the `-p` parameter indicates the minimum percent of bases that must have the quality specified by `-q` (here, `80`). As for the `-Q33` parameter, it indicates that the qualities present in the forth line of each read in the `.fastq` file were encoded by the Phred+33 encoding. After this filtering, we performed a second quality check using the same tools as before. The quality graphics seemed to have a better quality in average, as we wanted; but the k-mer contents sometimes got worse almost inexplicably.

2.3. Quality trimming

The reads we obtained after filtering, altough their good quality in average, still contained locally low quality reads. To remove the bases in our reads that had still a poor quality, we performed a quality trim. It eliminates low quality data at the beginning and end of a sequence. More precisely, data at the beginning and end will automatically be trimmed untilprocessable data is obtained. Once again, we used a FASTX-Toolkit[7] function called `fastq_quality_trimmer`. The parameter setting was as follows:

```
fastq_quality_trimmer -t 30 -l 20 -i <input> -o <output> -Q33
```

Where the `-t` parameter sets the quality threshold (nucleotides with lower quality will be trimmed from the end of the sequence), the `-l` parameter specifies the minimum length (sequences shorter than this after trimming will be discarded). As before, the `-Q33` option indicates that the qualities in the `.fastq` file were encoded using Phred+33 encoding.

For most of the samples, we were glad to notice that almost all the qualities were located in the green (good) area. However, some samples still presented strange k-mer peaks. After some tests with cutadapt [1], a tool used to clip specified adapters at the begining or at the end of each read, we decided nonetheless not to clip our sequences. Anyway, the command used to cut the sequences was the following:

```
cutadapt -b <adapter> -m 20 -o <output> <input>
```

[1] it was a tricky step done by Yuri, my tutor, so I did not really saw it in details

The adapter sequences were function of the sequencing machine, but the Illumina website provides a list of conventionnal adapters the differents machines are usually using. For us, the two most common adapter sequences were:

```
Illumina Multiplexing Adapter 1: GATCGGAAGAGCACACGTCT
Illumina Multiplexing Adapter 2: ACACTCTTTCCCTACACGACGCTCTTCCGATCT
```

2.4. Pairing

After these few steps, our data still had the same format, which means that we still had two files for each patient, one with forward reads, the other with backward reads. The problem is that our forward and backward reads did not received the same treatment during the previous steps: because of read deletions on both sides of our two files, the reads were not necessarily paired anymore. Practically, the n-th read of the forward file was probably not the "mate" of the n-th read of the backward file: they probably did not come from the same snippet of DNA. To settle this problem, we had to "sort" our files some way. To this aim, we use a Python script that found the remaining proper pairs in our two files by looking at their headers, and put aside the orphan reads both backward and forward sequences. Thus, the output consisted of three files: a file with all the paired forward reads (R1), another with all the paired backward reads (R2), and a last one with all the orphan reads created during the previous steps (S). Now, the n-th read of R1 and the n-th read of R2 were surely mates, which was mandatory to perform the mapping. Here is th Python script[8] we used to perform this pariring task:

```
#!/usr/bin/env python
import gzip
import sys

try:
    in1 = sys.argv[1]
    in2 = sys.argv[2]
except:
    print __doc__
    sys.exit(1)

try:
    separator = sys.argv[3]
except:
    separator = " "

class Fastq(object):
    def __init__(self, name, seq, name2, qual):
        self.name = name
        self.seq = seq
        self.name2 = name2
        self.qual = qual

    def getShortname(self, separator):
        self.temp = self.name.split(separator)
        del(self.temp[-1])
        return separator.join(self.temp)
```

```
    def getHeader(self):
        self.temp = self.name.split(" ")
        return self.temp[0] + " " + self.temp[1][0]

    def write_to_file(self, handle):
        handle.write(self.getHeader() + "\n")
        handle.write(self.seq + "\n")
        handle.write(self.name2 + "\n")
        handle.write(self.qual + "\n")

def myopen(infile, mode="r"):
    if infile.endswith(".gz"):
        return gzip.open(infile, mode=mode)
    else:
        return open(infile, mode=mode)

def fastq_parser(infile):
    with myopen(infile) as f:
        while True:
            name = f.readline().strip()
            if not name:
                break

            seq = f.readline().strip()
            name2 = f.readline().strip()
            qual = f.readline().strip()
            yield Fastq(name, seq, name2, qual)

if __name__ == "__main__":
    seq1_dict = {}
    seq2_dict = {}
    seq1 = fastq_parser(in1)
    seq2 = fastq_parser(in2)
    s1_finished = False
    s2_finished = False

    if in1.endswith('.gz'):
        outSuffix='.fastq.gz'
    else:
        outSuffix='.fastq'

    with myopen(in1 + "_1" + outSuffix, "w") as out1:
        with myopen(in2 + "_2" + outSuffix, "w") as out2:
            with myopen(in1 + "_S" + outSuffix, "w") as out3:
                while not (s1_finished and s2_finished):
                    try:
                        s1 = seq1.next()
                    except:
                        s1_finished = True
                    try:
                        s2 = seq2.next()
                    except:
                        s2_finished = True

                    if not s1_finished:
                        seq1_dict[s1.getShortname(separator)] = s1
                    if not s2_finished:
                        seq2_dict[s2.getShortname(separator)] = s2

                    if not s1_finished and s1.getShortname(separator) in seq2_dict:
                        seq1_dict[s1.getShortname(separator)].write_to_file(out1)
                        seq1_dict.pop(s1.getShortname(separator))
                        seq2_dict[s1.getShortname(separator)].write_to_file(out2)
                        seq2_dict.pop(s1.getShortname(separator))

                    if not s2_finished and s2.getShortname(separator) in seq1_dict:
                        seq2_dict[s2.getShortname(separator)].write_to_file(out2)
                        seq2_dict.pop(s2.getShortname(separator))
                        seq1_dict[s2.getShortname(separator)].write_to_file(out1)
                        seq1_dict.pop(s2.getShortname(separator))

                for r in seq1_dict.values():
                    r.write_to_file(out3)

                for r in seq2_dict.values():
                    r.write_to_file(out3)
```

3. Mapping

3.1. Reference genome indexing

Our reads were now prepared to be aligned with a reference genome. We chose the latest human genome released on the UCSC Genome Browser (hg38). However, this reference genome needed to be indexed before we could map it with the reads. We did this indexing with two different program, for some reasons that will become clearer in the next section. The first program was called Burrows-Wheeler Aligner (BWA[9]), the second was called Bowtie2[10].Both of them provide several tools for indexing and aligning reads against genomes. More precisely, the commands we used were:

```
bwa index hg38.fasta
bowtie2-build hg38.fasta hg38
```

Where `hg38.fasta` was the whole human genome downloaded *via* UCSC Genome Browser. These commands generated several files with several extensions (`.pac`, `.bwt`, `.sa`, `.amb`, `.ann` for BWA, `.X.bt2` and `.rev.X.bt2` for Bowtie2). We will not detail what the exactly contained, but they had to be useful for the next step.

3.2. Read-alignment

The read-alignment problem is, as we said, a difficult one. There exists many algorithm and many programs to perform this task, and it is not straightforward to assume one program is better than one another without trying both of them on a particular set of data. In our analysis, we chose to try three diefferent programs (BWA, Bowtie2 and STAR[11][2]) on one of our 30 samples, basically the first one, whose name was `as1017`. These programs output `.sam` files, which specify all the alignments of reads against the reference genome. for one patient, two `.sam` files had to be generated, one for the paired-end reads, the other for single reads. The commands for BWA were:

```
bwa mem <ref file> <r1> <r2> > <out>
bwa mem <ref file> <rs> > <out>
```

```
bowtie2 --phred33 -a -x  <ref prefix> -1 <r1> -2 <r2> -S <out>
bowtie2 --phred33 -a -x  <ref prefix> -U <rs> -S <out>
```

[2]I haven't used STAR actually, it was Nikolai - he worked with me on this project - who did the job.

Where <r1> and <r2> denote respectively forward reads and backward reads, and <rs> denotes single reads. However, the default parameters of Bowtie2 that we used did not allow multiple alignments for one reads which could make the performance comparison with BWA and STAR[11] less accurate. That is why we also run Bowtie2 with the option `-k 5`, which allows a maximum of five multimaps per read:

```
bowtie2 --phred33 -a -k 5 -x  <ref prefix> -1 <r1> -2 <r2> -S <out>
bowtie2 --phred33 -a -k 5 -x  <ref prefix> -U <rs> -S <out>
```

The process took quite a long time in each case, but then we had our data aligned, and viewable, as we will explain.

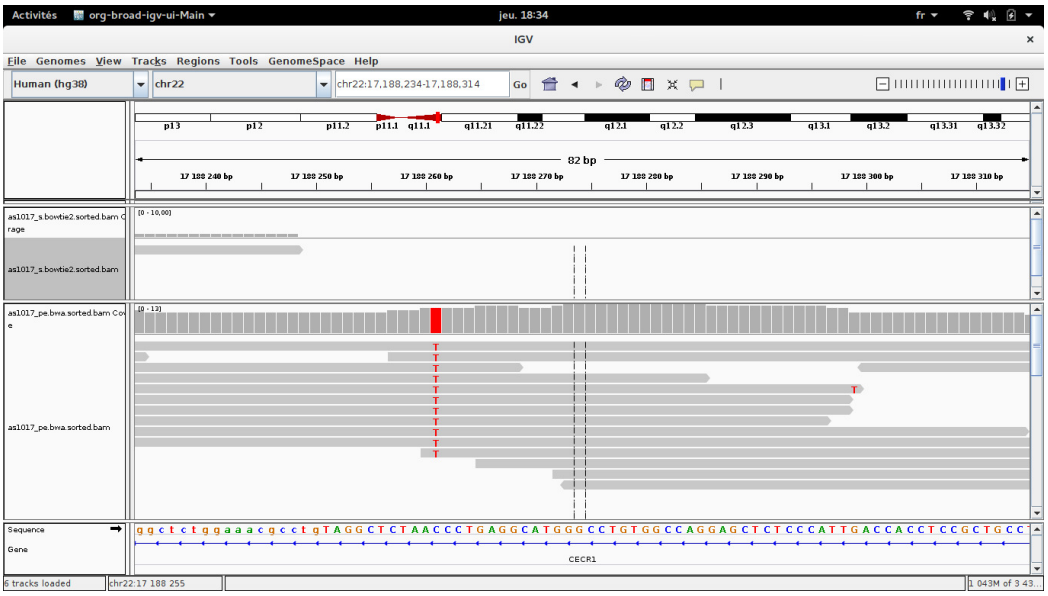
3.3. Visualizing the data

To see how the alignments looked like, we used a soft called Integrative Genomics Viewer (IGV[12, 13]), which works with java. But before running it, we had to do some additionnal tasks with our `.sam` files. Indeed, IGV doesn't use `.sam` files as they are; we needed to convert them into `.bam` file, then sort them and index them. These tasks could be done with a soft called SAMtools[14], as follows:

```
samtools view -b <sam file> > <bam file>
samtools sort -o <sorted output> <bam input>
samtools index <sorted bam input>
```

The last command outputed a `.bai` file, which was used with the `.bam` file by IGV. Here is a screen of what IGV can display. The red Ts that appear in most of the reads (and not in the reference) are probably a mutation.

Figure 3: Example of IGV interface



3.4. Benchmarking

After having mapped our reads with diffrent softwares, we had to choose the best one to peform the task on all the other files. To this aim, we collected some performance statistics for our three softwares, and we stored them in a tab.

We thus compared 3 softwares, but we used two different settings with bowtie2. The first (default), just returned the best alignment for each read; the second (`-k 5`), returned at most 5 possible alignments for each read, so that we could obtain multimaps for the reads.

Tools	N_{reads}	$N_{aligned}$	$\%_{aligned}$	N_{paired}	$\%_{paired}$
`bwa` (pe)	7145284	7145051	99.997	6969068	97.534
`bwa` (s)	451568	451484	99.981	0	0
bowtie2 (pe)	7145284	7133594	99.836	6898794	96.550
`bowtie2` (s)	451568	450601	99.786	0	0
`bowtie2 -k5` (pe)	7145284	7133956	99.841	6898876	96.551
`bowtie2 -k5` (s)	451568	450672	99.802	0	0
`STAR` (pe)	7145284	6977588	97.653	6977588	97.653
`STAR` (s)	451568	450418	99.745	0	0

(a) Number cleaned reads - number of aligned reads - number of proper pairs

Tools	$N_{!aligned}$	$\%_{!aligned}$	$N_{reversed}$	$\%_{reversed}$
`bwa` (pe)	233	0.003	3575710	50.045
`bwa` (s)	84	0.019	230289	51.007
bowtie2 (pe)	11690	0.164	3567153	50.005
`bowtie2` (s)	967	0.214	229628	50.960
`bowtie2 -k5` (pe)	11328	0.159	4000471	56.076
`bowtie2 -k5` (s)	896	0.198	274454	60.899
`STAR` (pe)	167696	2.347	3784966	54.245
`STAR` (s)	1150	0.255	254217	56.440

(b) Reads not aligned - Reversed reads

Tools	$N_{alignments}$	N_{supp}	$\%_{supp}$	$N_{!primary}$	$\%_{!primary}$
`bwa` (pe)	7150986	5935	0.083	0	0
`bwa` (s)	451872	388	0.086	0	0
bowtie2 (pe)	7133594	0	0	0	0
`bowtie2` (s)	450601	0	0	0	0
`bowtie2 -k5` (pe)	8000890	0	0	866934	10.835
`bowtie2 -k5` (s)	539496	0	0	88824	16.464
`STAR` (pe)	7569932	0	0	592344	7.825
`STAR` (s)	499056	0	0	48638	9.746

(c) Alignments - Supplementary alignments - Not primary alignments

Tools	$N_{\neq chr}$	$\%_{\neq chr}$
`bwa` (pe)	156070	2.182
`bwa` (s)	0	0
bowtie2 (pe)	145886	2.045
`bowtie2` (s)	0	0
`bowtie2 -k5` (pe)	146340	1.829
`bowtie2 -k5` (s)	0	0
`STAR` (pe)	0	0
`STAR` (s)	0	0

(d) Reads mapped in differents chromosomes

Table 1: Statistics of different read-alignment softwares

To fill these columns we used an alternative to the `samtools flagstat` command, maybe more complicated but also more accurate, to the extent that we could know exactly what we were counting. The commands were a combination of `samtools view -f -F` and `sort`, `uniq`, and `wc -l` commands. They are listed below.

- for N_{reads} :

```
grep -E -v "^@" <sam file> | cut -f 1 | sort | uniq | wc -l
```

- for $N_{aligned}$ with paired-end reads (pe) :

```
samtools view -f 128 -F 4 <file> | cut -f 1 | sort |
    uniq | wc -l
+ samtools view -f 64 -F 4 <file> | cut -f 1 | sort
   | uniq | wc -l.1
```

- for $N_{aligned}$ with single reads (s) :

```
samtools view -F 4 <file> | cut -f 1 | sort | uniq | wc -l
```

- for N_{paired} :

```
samtools view <file> -f 2 -F 4 | cut -f 1 | sort | uniq | wc -l
```

- for $N_{!aligned}$ with paired-end reads (pe) :

```
samtools view -f 132 <file> | cut -f 1 | sort | uniq
    | wc-l
+ samtools view -f 68 <file> | cut -f 1 | sort |
   uniq | wc-l
```

- for $N_{!aligned}$ with single reads (s) :

```
samtools view -f 4 <file> | cut -f 1 | sort | uniq | wc-l
```

- for $N_{reversed}$ with paired-end reads (pe) :

```
samtools view -f 80 -F 4 <file> | cut -f 1 | sort |
   uniq | wc -l
+ samtools view -f 144 -F 4 <file> |cut -f 1 | sort
   | uniq | wc -l
```

- for $N_{reversed}$ with single reads (s) :

```
samtools view -f 16 -F 4 <file> | cut -f 1 | sort | uniq | wc -l
```

- for $N_{alignments}$:

```
samtools view -F 4 <file> | wc -l
```

- for N_{supp} :

```
samtools view -f 2048 <file> | wc -l
```

- for $N_{!primary}$:

```
samtools view -f 256 <file> | wc -l
```

- for $N_{\neq chr}$:

```
samtools flagstat <file>
```

and then see the 12^{th} line.

These stats showed that BWA had the best performance on this example, so we decided to map all the remaining `.fastq` files using this tool.

3.5. BWA mapping

The scripts we used to map all our files were:

Listing 2: Mapping for paired-end reads

```
#!/bin/bash

for f in ../../clean/paired/*_1.cleaned.paired.fastq
do
        echo $f
        echo $(echo $f | sed 's/_R1/_R2/g')
        name=`basename $f .fastq`
        echo $name
        name=`basename $name .cleaned_R1`
        echo $name.sam
        bwa mem -t 8 hg38.fasta $f $(echo $f | sed 's/_1/_2/g') > $name.bwa.sam
done
```

Listing 3: Mapping for single reads

```
#!/bin/bash

for f in ../../clean/paired/*_S.cleaned.paired.fastq
do
        echo $f
        name=`basename $f .fastq`
        echo $name
        name=`basename $name .cleaned_R1`
        echo $name.sam
        bwa mem -t 3 hg38.fasta $f  > $name.bwa.sam
done
```

To be sure there had been no problems during the mapping, we made the same stats as before for all the `.sam` files obtained after the two mapping scripts.

FILE:	as1017_pe	at1016_pe	at5165_pe	au0412_pe
N_{reads}	7145284	15400210	20916372	15841744
$N_{aligned}$	7145051	15399591	20915284	15841185
$\%_{aligned}$	99.997	99.996	99.995	99.996
N_{paired}	6968938	14924242	20752004	15514796
$\%_{paired}$	97.532	96.909	99.214	97.936
$N_{!aligned}$	233	619	1088	559
$\%_{!aligned}$	0.003	0.004	0.005	0.004
$N_{reversed}$	3575710	7708115	10462842	7925526
$\%_{reversed}$	50.045	50.054	50.025	50.031
$N_{alignments}$	7150986	15414892	20925520	15852375
N_{supp}	5935	15301	10236	11190
$\%_{supp}$	0.083	0.099	0.049	0.071
$N_{!primary}$	0	0	0	0
$N_{\neq chr}$	156070	438872	127436	294768
$\%_{\neq chr}$	2.182	2.847	0.609	1.859

FILE:	au1907_pe	au2558_pe	av0650_pe	av9644_pe
N_{reads}	13014608	20351702	5449928	12368990
$N_{aligned}$	13014122	20350860	5449723	12368504
$\%_{aligned}$	99.996	99.996	99.996	99.996
N_{paired}	12704650	19966936	5379464	12243482
$\%_{p}aired$	97.618	98.109	98.707	98.985
$N_{!aligned}$	486	842	205	486
$\%_{!aligned}$	0.004	0.004	0.004	0.004
$N_{reversed}$	6513800	10184278	2727268	6187087
$\%_{reversed}$	50.052	50.043	50.044	50.023
$N_{alignments}$	13027082	20367926	5454981	12374035
N_{supp}	12960	17066	5258	5531
$\%_{supp}$	0.099	0.084	0.096	0.045
$N_{!primary}$	0	0	0	0
$N_{\neq chr}$	282592	330912	59552	94842
$\%_{\neq chr}$	2.169	1.625	1.092	0.766

FILE:	aw1665_pe	ax4064_pe	bb6509_pe	bc4701_pe
N_{reads}	23689518	15613558	14183902	20456218
$N_{aligned}$	23688427	15612442	14183358	20455405
$\%_{aligned}$	99.995	99.993	99.996	99.996
N_{paired}	23223668	15403668	13799930	19889032
$\%_{paired}$	98.034	98.656	97.293	97.227
$N_{!aligned}$	1091	1116	544	813
$\%_{!aligned}$	0.005	0.007	0.004	0.004
$N_{reversed}$	11853599	7811728	7098952	10238316
$\%_{reversed}$	50.040	50.035	50.051	50.052
$N_{alignments}$	23708554	15622595	14197254	20476248
N_{supp}	20127	10153	13896	20843
$\%_{supp}$	0.085	0.065	0.098	0.102
$N_{!primary}$	0	0	0	0
$N_{\neq chr}$	376342	157714	338268	484480
$\%_{\neq chr}$	1.587	1.010	2.383	2.366

FILE:	be5318_pe	bf0854_pe	bg0994_pe	bh5562_pe
N_{reads}	8377946	23332202	19884370	15773374
$N_{aligned}$	8377500	23331409	19883701	15772717
$\%_{aligned}$	99.995	99.997	99.997	99.996
N_{paired}	8254342	23021116	19678010	15609338
$\%_{paired}$	98.525	98.667	98.962	98.960
$N_{!aligned}$	446	793	669	657
$\%_{!aligned}$	0.005	0.003	0.003	0.004
$N_{reversed}$	4190572	11673962	9946808	7891196
$\%_{reversed}$	50.022	50.035	50.025	50.031
$N_{alignments}$	8381620	23347218	19893644	15781004
N_{supp}	4120	15809	9943	8287
$\%_{supp}$	0.049	0.068	0.050	0.053
$N_{!primary}$	0	0	0	0
$N_{\neq chr}$	106490	239372	136916	110212
$\%_{\neq chr}$	1.271	1.025	0.688	0.698

FILE:	bh6248_pe	bh7279_pe	bj9943_pe	bm8214_pe
N_{reads}	20635838	16243772	13339718	6880438
$N_{aligned}$	20634930	16243128	13339237	6880182
$\%_{aligned}$	99.996	99.996	99.996	99.996
N_{paired}	20492050	16095746	13234408	6746470
$\%_{paired}$	99.303	99.089	99.211	98.053
$N_{!aligned}$	908	644	481	256
$\%_{!aligned}$	0.004	0.004	0.004	0.004
$N_{reversed}$	10321019	8126500	6672655	3442431
$\%_{reversed}$	50.017	50.030	50.023	50.034
$N_{alignments}$	20641494	16252658	13345730	6884489
N_{supp}	6564	9530	6493	4307
$\%_{supp}$	0.032	0.059	0.049	0.063
$N_{!primary}$	0	0	0	0
$N_{\neq chr}$	72706	48170	64886	116402
$\%_{\neq chr}$	0.352	0.296	0.486	1.691

FILE:	bp9235_pe	bt5296_pe	bt7037_pe	bu7290_pe
N_{reads}	19905682	7506640	21439782	9726610
$N_{aligned}$	19904901	7506361	21439088	9726212
$\%_{aligned}$	99.996	99.996	99.997	99.996
N_{paired}	19439582	7393524	21194910	9513718
$\%_{paired}$	97.658	98.493	98.858	97.811
$N_{!aligned}$	781	279	694	398
$\%_{!aligned}$	0.004	0.004	0.003	0.004
$N_{reversed}$	9963829	3755064	10728170	4868392
$\%_{reversed}$	50.057	50.025	50.040	50.054
$N_{alignments}$	19927144	7510029	21455578	9736048
N_{supp}	22243	3668	16490	9836
$\%_{supp}$	0.112	0.049	0.077	0.101
$N_{!primary}$	0	0	0	0
$N_{\neq chr}$	402960	97106	169926	187004
$\%_{\neq chr}$	2.022	1.293	0.792	1.921

FILE:	bx2942_pe	bx3403_pe	bz1082_pe	ca8314_pe
N_{reads}	30049046	22764030	17344412	11521720
$N_{aligned}$	30048030	22763358	17343788	11521309
$\%_{aligned}$	99.997	99.997	99.996	99.996
N_{paired}	29490218	22611342	17216282	11372352
$\%_{paired}$	98.140	99.329	99.261	98.704
$N_{!aligned}$	1016	672	624	411
$\%_{!aligned}$	0.003	0.003	0.004	0.004
$N_{reversed}$	15036365	11385901	8674739	5766161
$\%_{reversed}$	50.041	50.019	50.016	50.048
$N_{alignments}$	30072352	22771955	17349746	11532207
N_{supp}	24322	8597	5958	10898
$\%_{supp}$	0.081	0.038	0.034	0.095
$N_{!primary}$	0	0	0	0
$N_{\neq chr}$	459492	115858	61564	94510
$\%_{\neq chr}$	1.528	0.509	0.355	0.820

FILE:	cc2190_pe	cc5110_pe
N_{reads}	10611350	8497910
$N_{aligned}$	10610959	8497612
$\%_{aligned}$	99.996	99.996
N_{paired}	10495044	8346880
$\%_{paired}$	98.904	98.223
$N_{!aligned}$	391	298
$\%_{!aligned}$	0.004	0.004
$N_{reversed}$	5308367	4251478
$\%_{reversed}$	50.027	50.031
$N_{alignments}$	10616406	8502759
N_{supp}	5447	5147
$\%_{supp}$	0.051	0.061
$N_{!primary}$	0	0
$N_{\neq chr}$	72316	135296
$\%_{\neq chr}$	0.681	1.591

	mean	median	min	max
N_{reads}	15608896	15693466	5449928	30049046
$N_{aligned}$	15608279	15692580	5449723	30048030
$\%_{aligned}$	99.9960	99.9962	99.9929	99.9970
N_{paired}	15365871	15459232	5379464	29490218
$\%_{paired}$	98.4124	98.5902	96.9093	99.3293
$N_{!aligned}$	617	622	205	1116
$\%_{!aligned}$	0.0040	0.0037	0.0030	0.0071
$N_{reversed}$	7809694	7851462	2727268	15036365
$\%_{reversed}$	50.0357	50.0346	50.0164	50.0572
$N_{alignments}$	15619151	15701800	5454981	30072352
N_{supp}	10872	9890	3668	24322
$\%_{supp}$	0.0698	0.0664	0.0318	0.1116
$N_{!primary}$	0	0	0	0
$N_{\neq chr}$	194434	136106	48170	484480
$\%_{\neq chr}$	1.2838	1.1811	0.2964	2.8470

Table 2: Satistics for BWA software on paired-end reads

We did the same kind of table for single reads. Then the files were almost ready for variant-calling.

4. Variant calling and annotation

The aim of variant calling is to automatically detect what we could manually see with IGV: gene sequence variations. These variations are stored in a `.vcf` file (for Variant Call Format). To perform this step, we used the Genome Analysis Tollkit (GATK[15, 16, 17]). Developed by the Data Science and Data Engineering group at the Broad Institute, the toolkit offers a wide variety of tools with a primary focus on variant discovery and genotyping. It is based on machine learning methods.

4.1. Reference genome preprocessing

First of all, the reference genome hg38 had to be preprocessed again, before using the GATK. We used a toolkit called Picard[18], which was also developped by the Broad Institute, to create a dictionnary of sequences for this genome, with the following command:

```
java -jar picard.jar CreateSequenceDictionary \
                     R= hg38.fasta \
                     O= hg38.dict
```

After this step, we obtained a file called `hg38.dict`. And then we also needed to index the genome with SAMtools, with the command:

```
samtools faidx hg38.fasta
```

Which outputed a file called `hg38.fasta.fai`

4.2. `.sam` files preprocessing

Then, the `.sam` files outputed by BWA also needed some modifications. Indeed, the GATK took only one output per patient, which means we had to merge, for each patient, the paired-end `.sam` file and the single `.sam` file. We used SAMtools for this step:

```
samtools merge <output> <input paired-end> <input singles>
```

The corresponding bash script was:

Listing 4: Automatic merging

```
#!/bin/bash
for f in ../mapped/*_pe.sam
do
        echo $f
        name_single=$(echo $f | sed 's/_pe/_s/')
        name_merged=$(echo $f | sed 's/_pe//' | sed 's/mapped/merged/')
        echo $name_single
        echo $name_merged
        samtools merge $name_merged $f $name_single
done
```

We obtained for each patient a condensed `.sam` file, which contained both paired and single alignments. But another problem was that the GATK took `.bam` files (binary format for storing sequence data) as an output, not `.sam` files. Thus we had to convert our `.sam` files to this format, also with SAMtools:

```
samtools view -b <sam file> > <bam file>
```

The bash script was also straightforward:

Listing 5: Automatic conversion

```
#!/bin/bash
for f in *.sam
do
        name_bam=$(echo $f | sed 's/sam/bam/')
        echo $name_bam
        samtools view -b $f > $name_bam
done
```

Then the `.bam` had to be a kind of prepared, by ading a group header to them, sorting them, and finally indexing them. These tasks were performed with Picard tools, as follows:

```
java -jar picard.jar AddOrReplaceReadGroups \
                     I=<bam> \
                     O=<grouped bam> \
                     RGLB=lib1 \
                     RGPL=illumina \
                     RGPU=unit1 \
                     RGSM=20

java -jar picard.jar SortSam \
                     INPUT=<grouped bam> \
                     OUTPUT=<grouped sorted bam> \
                     SORT_ORDER=coordinate

java -jar picard.jar BuildBamIndex \
                     INPUT=<grouped sorted bam>
```

The corresponding script was:

Listing 6: Automatic .bam preprocessing

```
#!/bin/bash
for f in ../bwa/merged/*.bam
do
```

```
        echo $f
        name_grouped=$(echo $f | sed 's/.bam/.grouped.bam/' | sed 's/merged/grouped/')
        echo $name_grouped

        java -jar picard.jar AddOrReplaceReadGroups \
                             I=$f \
                             O=$name_grouped \
                        RGLB=lib1 \
                             RGPL=illumina \
                             RGPU=unit1 \
                             RGSM=20

        name_sorted=$(echo $name_grouped | sed 's/.bam/.sorted.bam/' | sed 's/grouped/sorted/')
        echo $name_sorted

        java -jar picard.jar SortSam \
                             INPUT=$name_grouped \
                             OUTPUT=$name_sorted \
                             SORT_ORDER=coordinate

        java -jar picard.jar BuildBamIndex \
                             INPUT=$name_sorted
done
```

4.3. Variant calling using the GATK

The tool we used to perform variant calling was HaplotypeCaller. The script used to configure this tool was:

Listing 7: Variant calling script

```
#!/bin/sh
java -jar GenomeAnalysisTK.jar \
          -nct 30 \
          -T HaplotypeCaller \
          -R hg38.fasta \
          -I $1 \
          --genotyping_mode DISCOVERY \
          -stand_emit_conf 10 \
          -stand_call_conf 30 \
          -o $2
```

The first argument passed to this script was the sorted, grouped and indexed `.bam` file we got at the end ofthe preprocessing tasks. The second argument was the output `.vcf` file. Notice that the HaplotypeCaller tool outputed a second file in addition to the `.vcf`; its extension was `.vcf.idx`. So, to automatize this call, we ran a second bash script:

Listing 8: Automatic variant calling

```
#!/bin/bash
for f in ../bwa/sorted/*.bam
do
        output=$(echo $f | sed 's/sorted/GATK/' | sed 's/.grouped.sorted.bam/.vcf/')
        echo $output
        ./call.sh $f $output
done
```

4.4. `.vcf` merging

Before annotating the `.vcf` files, we had to group them into two different `.vcf` files, according to the disease they were associated with. Here is a table that sumarize which identifier corresponds to which disease:

Disease	Sample ID	Sample sequencing ID	Initial DNA concentration
PD	PD1	CC5110	65
PD	PD2	CC2190	126
PD	PD3	CA8314	33.1
PD	PD4	BZ1082	43.9
PD	PD5	BX3403	68.5
PD	PD6	BX2942	201
PD	PD7	BU7290	129
PD	PD8	BT7037	75
PD	PD9	BT5296	10.5
PD	PD10	BP9235	17.2
PD	PD11	BM8214	10.4
PD	PD12	BJ9943	39.9
PD	PD13	BH7279	34.4
PD	PD14	BH6248	68
PD	PD15	BH5562	62
HCM	cm1	BG0994	21.4
HCM	cm2	BF0854	24.6
HCM	cm3	BE5318	13.5
HCM	cm4	BC4701	37.3
HCM	cm5	BB6509	146
HCM	cm6	AX4064	66.5
HCM	cm7	AW1665	21.8
HCM	cm8	AV9644	76
HCM	cm9	AV0650	51
HCM	cm10	AU2558	40.5
HCM	cm11	AU1907	31.2
HCM	cm12	AU0412	52.5
HCM	cm13	AT5165	10.2
HCM	cm14	AT1016	27
HCM	cm15	AS1017	53

Table 3: Patients' IDs and diseases

Consequently, we had to create two differents merged `.vcf` files, one for HCM disease and the other for PD. The merging task was performed with a tool from the VCFtools package, called `vcf-merge`. It also involved to compress the input `.vcf` files with `bgzip`, and index them with `tabix`, a generic

indexer for TAB-delimited genome position files, included in the SAMtools package:

Listing 9: Automatic merging

```
#!/bin/bash
for f in *.vcf
do
        bgzip $f > $f.gz
        tabix -p vcf $f.gz
        vcf-merge *.vcf.gz > out.vcf
done
```

The `bgzip` command compressed each `.vcf` file, creating `.vcf.gz` files. Then, the `tabix` command took the zipped files and generated some `.vcf.gz.tbi` files. These were used during the merger. Of course, we took care to create two different directories (one for PD, one for HCM), and we ran this script in them separately, so that we obtained one merged file for each disease[3].

4.5. Annotation

For the annotation, we used a tool called Annovar[19]

Listing 10: Automatic merging

```
#!/bin/sh

table_annovar.pl $1 humandb/ \
-buildver hg38 \
-out $2 \
-remove \
-protocol knownGene,\
          cytoBand,\
          genomicSuperDups,\
          esp6500siv2_all,\
          1000g2015aug_all,\
          1000g2015aug_afr,\
          1000g2015aug_eas,\
          1000g2015aug_eur,\
          avsnp147,\
          dbnsfp30a,\
          clinvar_20160302,\
          cosmic70,\
          exac03 \
-operation g,r,r,f,f,f,f,f,f,f,f,f,f \
-nastring . \
-vcfinput
```

This script took as arguments the merged `.vcf` files (one for PD, one for HCM), and the prefix of the output. It outputed several files, including one `.vcf` file and one `.txt` file, wich were basically identical. But we used the

[3]back then, we had actually some problems with the headers of our files. Indeed, the identifier of the files directly written in the `.vcf` was always `20`, due to the default parametrization of the Picard tool `AddOrReplaceReadGroups` (`RGSM=20`). We had to change these identifiers manually in the merged files. We chose to call them with the format `<sampe ID>_<sample sequencing ID>` to avoid any ambiguity.

`.txt` files for further analysis on a spreadsheet. Actually, the annotation found for each mutation detected, some additionnal informations like the type of this mutation (exonic, intronic, non-coding RNA...), the impact of the mutation on the protein synthesis (synonymous or non-synonymous mutation...), the name of the gene involved (according to the UCSC database), and a description of the gene's function. We thus could make several statistics on these mutation:

	HC	HCM (%)	PD	PD (%)
N_{lines}	495899.0		505335.0	
N_{exonic}	30550.0	6.1605	31137.0	6.1617
$N_{exonic;splicing}$	54.0	0.0109	50.0	0.0099
$N_{intronic}$	252333.0	50.8840	253637.0	50.1919
$N_{intergenic}$	138840.0	27.9976	144511.0	28.5971
N_{UTR3}	11809.0	2.3813	11823.0	2.3396
N_{UTR5}	5459.0	1.1008	5667.0	1.1214
$N_{UTR5;UTR3}$	71.0	0.0143	60.0	0.0119
$N_{ncRNA;exonic}$	5691.0	1.1476	6042.0	1.1956
$N_{ncRNA;exonic;splicing}$	1.0	0.0002	1.0	0.0002
$N_{ncRNA;splicing}$	22.0	0.0044	21.0	0.0042
$N_{ncRNA;intronic}$	40444.0	8.1557	41265.0	8.1659
$N_{ncRNA;UTR5}$	1.0	0.0002	0.0	0.0000
$N_{upstream}$	5629.0	1.1351	5889.0	1.1654
$N_{downstream}$	4227.0	0.8524	4473.0	0.8852
$N_{upstream;downstream}$	473.0	0.0954	466.0	0.0922
$N_{splicing}$	295.0	0.0595	293.0	0.0580
Total exonic	36296	7.3192	37230	7.3674
Total exonic(;splicing)	30604		31187	
Total	495899	100.0000	505335	100.0000
$N_{nonsynonymous}$	15788.0	3.1837	16015.0	3.1692
$N_{synonymous}$	13072.0	2.6360	13267.0	2.6254
$N_{\neq genes}$	40693.0		41112.0	
$N_{\neq exonicgenes}$	7853.0		7930.0	
$N_{\neq !synonymousgenes}$	40471.0		40885.0	

Table 4: Annotation stats

To obtain these stats, we used the `grep` command, which led us to the

following script:

Listing 11: Stats on mutation types

```
#!/bin/bash
for f in *_annotation.hg38_multianno.txt
do
echo $f
echo "NUMBER OF LINES IN TXT    " $(($(cat $f | wc -l) - 1))
echo "EXONIC     " $(cat $f | cut -f 6 | grep "exonic" | grep -v ";" | grep -v "_" | wc -l)
echo "EXONIC & SPLICING " $(cat $f | cut -f 6 | grep "exonic;splicing" | grep -v "ncRNA" |
wc -l)
echo "INTRONIC   " $(cat $f | cut -f 6 | grep "intronic" | grep "intronic" | grep -v "_" | wc -l)
echo "INTERGENIC         " $(cat $f | cut -f 6 | grep "intergenic" | wc -l)
echo "UTR3       " $(cat  $f | cut -f 6 | grep "UTR3" | grep -v ";" | wc -l)
echo "UTR5       " $(cat $f | cut -f 6 | grep "UTR5" | grep -v ";" | grep -v "_" | wc -l)
echo "UTR3 & UTR5        " $(cat $f | cut -f 6 | grep "UTR5;UTR3" | wc -l)
echo "NCRNA EXONIC       " $(cat $f | cut -f 6 | grep "ncRNA_exonic" | grep -v ";" | wc -l)
echo "NCRNA EXONIC & SPLICING    " $(cat $f | cut -f 6 | grep "ncRNA_exonic;splicing" | wc -l)
echo "NCRNA SPLICING     " $(cat $f | cut -f 6 | grep "ncRNA_splicing" | wc -l)
echo "NCRNA INTRONIC     " $(cat $f | cut -f 6 | grep "ncRNA_intronic" | wc -l)
echo "NCRNA UTR5         " $(cat $f | cut -f 6 | grep "ncRNA_UTR5" | wc -l)
echo "UPSTREAM   " $(cat $f | cut -f 6 | grep "upstream" | grep -v ";" | wc -l)
echo "DOWNSTREAM         " $(cat $f | cut -f 6 | grep "downstream" | grep -v ";" | wc -l)
echo "UPSTREAM & DOWNSTREAM      " $(cat $f | cut -f 6 | grep "upstream;downstream" | wc -l)
echo "SPLICING   " $(cat $f | cut -f 6 | grep "splicing" | grep -v "_" | grep -v ";" | wc -l)
echo "NON SYNONYMOUS     " $(cat $f | cut -f 9 | grep "nonsynonymous" | wc -l)
echo "SYNONYMOUS         " $(cat $f | cut -f 9 | grep "synonymous" | grep -v "nonsyn" | wc -l)
```

Listing 12: Stats on distinct genes

```
#!/bin/bash

rm stats_genes.txt
for g in annotated/*_annotation.hg38_multianno.txt
do
echo $g
f=$(echo $g | sed 's/annotated//g' | sed 's/_annotation.hg38_multianno.txt//g' | sed 's,/,,g')
echo $f
cat $g | cut -f 7 | tail -n +2 | sort | uniq > $f.genes.txt
cat $g | cut -f 6-7 | grep "exonic" | grep -v "_" | cut -f 2 | sort | uniq > $f.exonic_genes.txt
cat $g | cut -f 7-9 | grep "nonsynonymous" | cut -f 1 | sort | uniq > $f.nonsynonymous_genes.txt
cat $g | cut -f 7-9 | grep -v "synonymous" | cut -f 1 | sort | uniq >> $f.nonsynonymous_genes.txt
sed 's/ /\n/g' $f.genes.txt
sed 's/ /\n/g' $f.exonic_genes.txt
sed 's/ /\n/g' $f.nonsynonymous_genes.txt
sort -u $f.nonsynonymous_genes.txt > temp.txt
mv temp.txt $f.nonsynonymous_genes.txt
echo $f >> stats_genes.txt
echo "NUMBER OF GENES    " $(cat $f.genes.txt | wc -l) >> stats_genes.txt
echo "NUMBER OF EXONIC GENES     " $(cat $f.exonic_genes.txt | wc -l) >> stats_genes.txt
echo "NUMBER OF NONSYNONYMOUS GENES      " $(cat $f.nonsynonymous_genes.txt | wc -l) >> stats_genes.txt
done
```

We also made different subtables which contained critical mutations, that's to say, exonic and non-synonymous mutations (that can deeply modify the related protein), and mutations associated with particular genes which are known to cause PD or HCM. The list of these genes is detailed below.

PD	HCM	
SNCA	MYH7	MTTI
Parkin	MYBPC3	MTTK
PINK1	TNNT2	MTTQ
DJ-1	TNNI3	MYL2
LRRK2	ACTC1	TTR
ATP13A2	CAV3	MYL3
PLA2G6	GLA	PRKAG2
FBX07	LAMP2	TNNC1
VPS35	MTTG	TPMI

Table 5: List of critical genes for HCM and PD

In particular, we found that genes LRRK2, VPS35, ATP13A2, PINK1, PLA2G6 were altered by non-synonymous mutations, respectively with 8, 1, 5, 4, 1 occurences. Similarly for HCM, genes MYBPC3, MYL2, MYH7, ACTC1, TTR, CAV3, GLA, LAMP2 appeared with non-synonymous mutations with respectively 5, 1, 4, 1, 1, 1, 1, 1 occurences.

	Gene	$N_{nonsynonymous}$
PD	LRRK2	8
	VPS35	1
	ATP13A2	5
	PINK1	4
	PLA2G6	1
HCM	MYBPC3	5
	MYL2	1
	MYH7	4
	ACTC1	1
	TTR	1
	CAV3	1
	GLA	1
	LAMP2	1

Table 6: Count of non-synonymous mutations on critival genes

We also detected genes that did not appear in the table below, but contained in their commentaries a mention of "Parkinson's disease" or "Hyper-

trophic cardiomyopathy". We took apart these particular genes, which are listed below.

PD	HCM		
TRPM7	ILK	DSP	CALR3
BDNF	CSRP3	NEBL	ABCC9
NDUFV2	MYH6	DSC2	DSG2
UCHL1	MYLK2	GPD1L	PKP2
PARK2	RP11-138H11.1	DTNA	JPH2
	SCO2	CTD-2587H24.4[1]	TPM1
	MYPN	VCL	NEXN
	ANKRD1	GATAD1	RBM20
	CRYAB	LMNA	TTN
	TTN-AS1	PDLIM3	TNNT2
	DES	ACTN2	RYR2

Table 7: List of other genes linked with HCM or PD

[1]associated with TNNI3

	Gene	$N_{nonsynonymous}$
PD	TRPM7	3
	BDNF	1
	NDUFV2	2
	UCHL1	3
	PARK2	3
HC	CSRP	1
	MYH6	5
	MYLK2	1
	SCO2	1
	MYPN	9
	ANKRD1	8
	CRYAB	1
	DSP	11
	NEBL	4
	DSC2	2
	GPD1L	1
	DTNA	2
	VCL	3
	GATAD1	1
	LMNA	6
	ACTN2	2
	CALR3	1
	DSG2	5
	PKP2	5
	JPH2	3
	NEXN	2
	RBM20	2
	TTN	109
	RYR2	7

Table 8: Count of non-synonymous mutations on other genes linked with PD or HCM

The genes that do not appear in Table 8, but do in Table 7, have just no non-synonymous mutations.

4.6. Mapping with the KEGG databases

KEGG[20] (Kyoto Encyclopedia of Genes and Genomes) is a collection of databases dealing with genomes, biological pathways, diseases, drugs, and chemical substances. The KEGG PATHWAY database, the wiring diagram database, is the core of the KEGG[20] resource. It is a collection of pathway maps integrating many entities including genes, proteins, RNAs, chemical compounds, glycans, and chemical reactions, as well as disease genes and drug targets, which are stored as individual entries in the other databases of KEGG[20]. The metabolism section contains aesthetically drawn global maps showing an overall picture of metabolism, in addition to regular metabolic pathway maps. We used this section to visualize the role of the critical genes we found during our study, regarding the whole metabolism. What we obtained for PD anfd HCM is showed below:

Figure 4: Metabolism diagram for PD

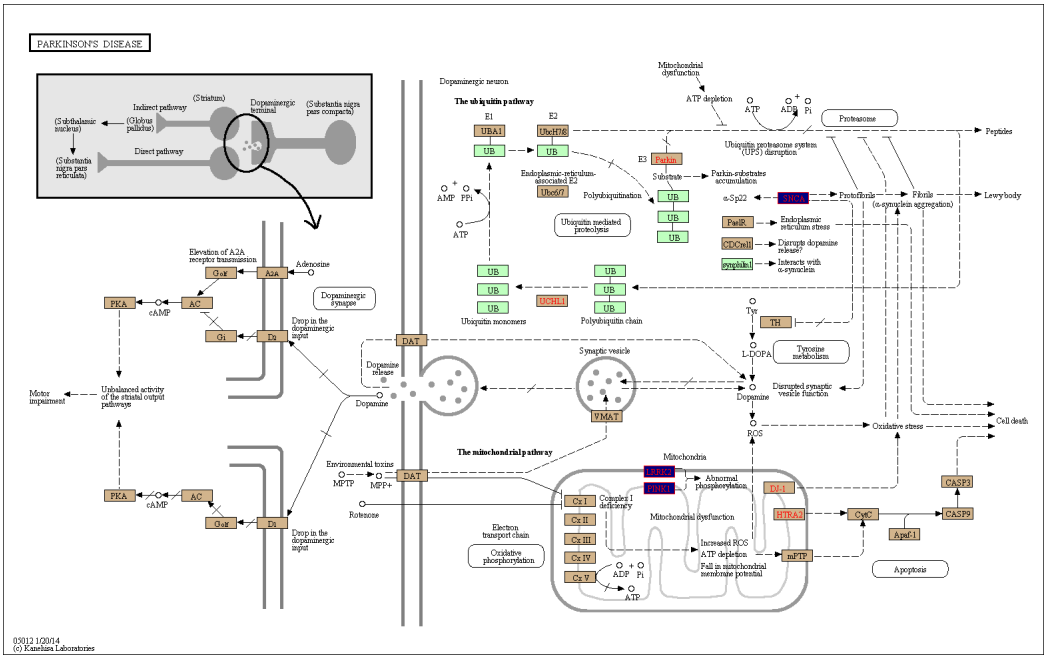
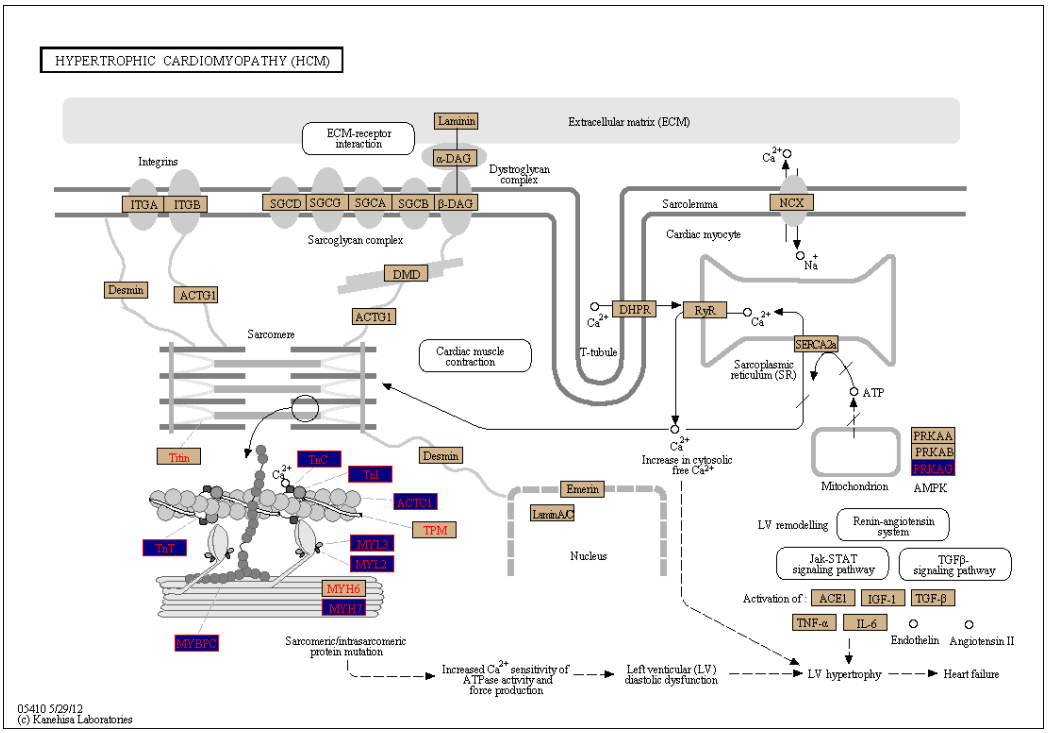


Figure 5: Metabolism diagram for HCM



In these diagrams, navy blue labels refer to the listed "critical genes" whereas tan labels refer to genes which were altered by a mutation that was not synonymous (non-synonymous but also other types...).

References

[1] NINDS, Parkinson's Disease Information Page, `http://www.ninds.nih.gov/disorders/parkinsons_disease`, [Online; accessed 30-June-2016; retrieved 10-July-2016].

[2] B. J. Maron, Hypertrophic cardiomyopathy: a systematic review, JAMA 287 (10) (2002) 1308–1320.

[3] P. Richardson, W. McKenna, M. Bristow, B. Maisch, B. Mautner, J. O'Connell, E. Olsen, G. Thiene, J. Goodwin, I. Gyarfas, I. Martin, P. Nordet, Report of the 1995 World Health Organization/International Society and Federation of Cardiology Task Force on the Definition and Classification of cardiomyopathies, Circulation 93 (5) (1996) 841–842.

[4] M. V. Sherrid, F. A. Chaudhry, D. G. Swistel, Obstructive hypertrophic cardiomyopathy: echocardiography, pathophysiology, and the continuing evolution of surgery for obstruction, Ann. Thorac. Surg. 75 (2) (2003) 620–632.

[5] E. D. Wigle, Z. Sasson, M. A. Henderson, T. D. Ruddy, J. Fulop, H. Rakowski, W. G. Williams, Hypertrophic cardiomyopathy. The importance of the site and the extent of hypertrophy. A review, Prog Cardiovasc Dis 28 (1) (1985) 1–83.

[6] E. D. Wigle, H. Rakowski, B. P. Kimball, W. G. Williams, Hypertrophic cardiomyopathy. Clinical spectrum and treatment, Circulation 92 (7) (1995) 1680–1692.

[7] H. Lab, FASTX-Tollkit, `http://hannonlab.cshl.edu/fastx_toolkit/`, [Online; accessed 16-August-2016].

[8] E. Normandeau, GitHub repository for the pairing script (fastqCombinePairedEnd.py), `https://github.com/enormandeau/Scripts`, [Online; accessed 16-July-2016].

[9] H. Li, R. Durbin, Fast and accurate long-read alignment with burrows–wheeler transform (2010).

[10] B. Langmead, S. L. Salzberg, Fast gapped-read alignment with bowtie 2, Nat Meth 9 (4) (2012) 357–359, brief Communication.
URL `http://dx.doi.org/10.1038/nmeth.1923`

[11] A. Dobin, C. A. Davis, F. Schlesinger, J. Drenkow, C. Zaleski, S. Jha, P. Batut, M. Chaisson, T. R. Gingeras, Star: ultrafast universal rna-seq aligner, Bioinformatics 29 (1) (2013) 15–21, 23104886[pmid]. doi:10.1093/bioinformatics/bts635.
URL `http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3530905/`

[12] J. T. Robinson, H. Thorvaldsdottir, W. Winckler, M. Guttman, E. S. Lander, G. Getz, J. P. Mesirov, Integrative genomics viewer, Nat Biotech 29 (1) (2011) 24–26. doi:10.1038/nbt.1754.
URL `http://dx.doi.org/10.1038/nbt.1754`

[13] H. Thorvaldsdóttir, J. T. Robinson, J. P. Mesirov, Integrative genomics viewer (igv): high-performance genomics data visualization

and exploration, Briefings in Bioinformatics 14 (2) (2013) 178–192. arXiv:http://bib.oxfordjournals.org/content/14/2/178.full.pdf+html, doi:10.1093/bib/bbs017.
URL http://bib.oxfordjournals.org/content/14/2/178.abstract

[14] H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Marth, G. Abecasis, R. Durbin, . G. P. D. P. Subgroup, The sequence alignment/map format and samtools, Bioinformatics 25 (16) (2009) 2078–2079, btp352[PII]. doi:10.1093/bioinformatics/btp352.
URL http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2723002/

[15] M. A. DePristo, E. Banks, R. Poplin, K. V. Garimella, J. R. Maguire, C. Hartl, A. A. Philippakis, G. del Angel, M. A. Rivas, M. Hanna, A. McKenna, T. J. Fennell, A. M. Kernytsky, A. Y. Sivachenko, K. Cibulskis, S. B. Gabriel, D. Altshuler, M. J. Daly, A framework for variation discovery and genotyping using next-generation dna sequencing data, Nat Genet 43 (5) (2011) 491–498. doi:10.1038/ng.806.
URL http://dx.doi.org/10.1038/ng.806

[16] A. McKenna, M. Hanna, E. Banks, A. Sivachenko, K. Cibulskis, A. Kernytsky, K. Garimella, D. Altshuler, S. Gabriel, M. Daly, M. A. DePristo, The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data, Genome Res. 20 (9) (2010) 1297–1303.

[17] G. A. Van der Auwera, M. O. Carneiro, C. Hartl, R. Poplin, G. del Angel, A. Levy-Moonshine, T. Jordan, K. Shakir, D. Roazen, J. Thibault, E. Banks, K. V. Garimella, D. Altshuler, S. Gabriel, M. A. DePristo, From FastQ Data to High-Confidence Variant Calls: The Genome Analysis Toolkit Best Practices Pipeline, John Wiley & Sons, Inc., 2002. doi:10.1002/0471250953.bi1110s43.
URL http://dx.doi.org/10.1002/0471250953.bi1110s43

[18] B. Institute, Picard, http://picard.sourceforge.net, [Online; accessed 15-August-2016].

[19] K. Wang, M. Li, H. Hakonarson, Annovar: functional annotation of genetic variants from high-throughput sequencing data, Nucleic Acids Research 38 (16) (2010) e164. arXiv:http://nar.oxfordjournals.org/content/38/16/e164.full.pdf+html,

doi:10.1093/nar/gkq603.
URL http://nar.oxfordjournals.org/content/38/16/e164.abstract

[20] M. Kanehisa, S. Goto, Kegg: Kyoto encyclopedia of genes and genomes, Nucleic Acids Res 28 (1) (2000) 27–30, gkd027[PII].
URL http://www.ncbi.nlm.nih.gov/pmc/articles/PMC102409/