Detection of aberrant splicing in FUS knock outs by means of RNA-seq data

Manuel Ruch

Research practical supervised by Martino Colombo

University of Bern, 18.09.15

Introduction

Amyotrophic lateral sclerosis (ALS) is a neurodegenerative disease, causing a progressive loss of motor neurons [1]. In the most severe state, it causes paralysis and eventually leads to death. FUS is one of the genes that is often found to be mutated in ALS patients. Despite these findings only little is known about how the mutations in FUS are causing the disease.
Our group has found an association of FUS with the minor spliceosome, a splicing complex different from the canonical spliceosome. A decrease in the expression levels of the minor spliceosome has already been observed in ALS as well as in spinal muscular atrophy (SMA), an other neurodegenerative disorder affecting the motor neurons [2]. Furthermore our group has confirmed a decreased splicing efficiency of reporter genes in FUS knock downs by means of qPCR. In such analyses the ratio of unspliced to spliced RNA fragments was estimated by using appropriate primers. In that manner only a subset of splice sites could be investigated. Thus the question is remaining, whether the deletion of FUS has a brought effect on splicing in neurons. In order to address this question one has to investigate the entire transcriptome by means of RNA-seq data of FUS KOs and wild type cells. Such data has previously been acquired. Thus the aim of this project was to analyse this data for differences in splicing efficiency.

Data

The data that was used in this project was obtained from SH-SY5Y neuroblastoma cells, which is commonly used as a model for neurons. Three distinct KO clones were generated. The transcriptome of four replicate samples from every KO as well as the transcriptome of four samples from the wild type were sequenced and mapped to the latest version of the human genome (GRCh38). Thus the analysis was performed with the corresponding 16 bam files. Such files are a special compressed binary version of SAM files, which allow fast random access to every read [3]. Even though only the SAM files are human-readable. They consist of a tab delimited line for each aligned read. Eleven sections are present in every line, among which a bit wise flag which among other information contains the read orientation, the read reference name which in our case indicates the chromosome name of the read reference, the read reference start, the mapping quality and the CIGAR string containing the information about the mapping of the read. Those are the sections that were used in the analysis.

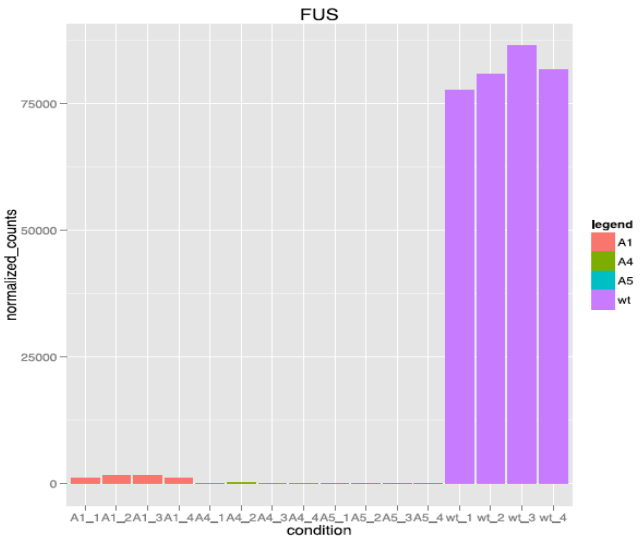

Figure 1: coverage of FUS transcripts in the analysed samples. Despite the knock out a small fraction of coverage was remaining in all of the three clones.

Interestingly a small amount of FUS RNA could be found in every sample from the KO clones (fig. 1). This might be because the KO was created by introducing an internal stop codon, which might allow a small

amount of transcription by read through. Alternatively the cell lines regarded as clones could be heterogeneous, meaning that it emerged from several cells, including some wild type cells. Nevertheless the expression level of FUS is drastically reduced in all of the KO clones, which allows an investigation of the implication of absence of FUS on splicing efficiency.

Methods

Detection of splice sites and counting of intron coverage

The provided bam-file could be parsed by using the pysam package of python. This allows to access the fields of every read, including reference start as well as a representation of the CIGAR string. The CIGAR string contains information about the way that a read matches to the reference genome. In particular it contains the number of non-matching nucleotides on the reference genome, which usually indicates the presence of an intron with that length. Thus in a first step every read in the file was checked for spanning an intron. By this a python dictionary containing all splice sites in the sample as keys, mapping to the number of reads across this site could be established. A string containing the chromosome, the start and the stop as well as the read orientation (eg chr8_42396681_42398711_+) was used as a key. In this step only reads with a minimal mapping quality of 40 were used, which excludes multi-mapped reads.

In order to find out how splicing could be affected by the absence of FUS every detected site was checked for reads that map to the intron. This was addressed by firs using the fetch function of the pysam module, which allows to get all reads overlapping the site. Reads overlapping the intros with less than 10 nucleotides were excluded. In that way the coverage of the intron could be estimated, which corresponds to the amount of unspliced pre-mRNA in the sample (fig. 2a).

However this step of the analysis can be confounded by alternatively spliced sites (fig. 2b). In such sites it can occur that a read mapping to the intron actually represents an exon of an alternative splice variant. Thus the reads had to be checked, if they contain an intron by them self. Such reads probably arise from an exon of an alternatively spliced mRNA and thus can not be counted. Moreover the coverage of an intron that is subject to alternative splicing can not be estimated since a read mapping to this intron could as well be a read mapping to an exon of an other splice variant, even if the read in question does not span a splice site it self. The number of such ambiguous reads was counted in order to finally apply a threshold of ambiguous reads per counted read. The analysis was run twice once with a threshold of 0 and once with 0.05 ambiguous reads per intron count. The coverage of sites with a larger ratio of ambiguous reads to normal reads than the threshold were set to 'Na' The output of the python script was written to a tab delimited file with site per line containing the normalisation counts and the coverage of the site. The estimation of normalisation counts and the coverage was performed individually with every bam file, resulting in different output files.

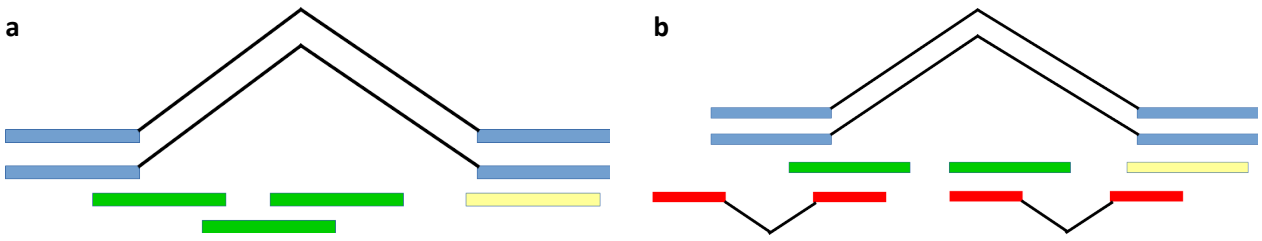

Figure 2: a) Schema of a site without alternative splicing, in blue the reads mapping across the splice site, used for detection of the site, their number corresponds to the normalisation counts, in green the reads mapping to the intron indicating an unspliced RNA fragment, their number corresponds to the counts, in yellow a read with a small overlap, that is excluded from the analysis. **b)** Schema of a site that is subject to alternative splicing, in red the reads mapping to the intron that contain an other intron by them self. Such reads are indication of alternative splicing, if the threshold of red to green reads exceeds the threshold the number of counts would be labelled with 'Na'.

Annotation and concatenation

Upon sorting the output files using bash, they were compared to a gtf file containing all know minor splice sites in the human genome, again with python. In the output every site was tagged with either canonical or minor, if present in the gtf file, sites containing 'Na' for the count were omitted in this step. Two output files were created one containing the number of reads across the intron, the other containing other containing the number of reads inside the splice site. This was done for simplifying the further down stream processing.

Now all output files had to be concatenated resulting in only two files, again one for the counts across the other for the counts from inside the site. This was done as follows by using bash. First a sorted list without duplicates of all sites present in all files was created then the counts were added successively by joining the list with the corresponding output files. The corresponding bash script can be found in the appendix.

Statistical analysis

The statistical analysis was done in R using the DESeq2 package [4]. Before putting the data through the DESeq2 pipeline a threshold for a minimal number of counts of reads across and reads from within the splice site for each of the four different KO clones was applied. Sites with fewer counts in one or more sample were excluded from the analysis. This was done with a thresholds of 20 reads per clone or wild type, respectively a minimal average of 5 reads per KO clone or wild type.

The analysis was done on the counts from within the site and using the counts across the site as site wise normalization factors. By doing so, the expression level of each considered site was always analysed with respect to the expression level of the corresponding splice site. Therefore we could compensate for differences in expression and focus exclusively on differences in splicing efficiency. The results were sorted by the p-value of every site and exported in a csv file.

Quality assessment

For the quality assessment of the analysis several measures were used. Firs of all a PCA plot of the resulting differential splicing efficiency of the samples. This was done again with the DESeq2 package. Then the ten most significant sites were further tested for variation and distribution of their coverage. A simple plot with the average number and standard deviation of counts and normalisation counts per clone was produced. Further more the distribution of the coverage over the splice site was of particular interest. This coverage distribution was calculated with a python script, which takes as an input a file with the sites in question, an other file with the file names of the bam files of the samples in question of course the bam files it self. It would then fetch all reads from one bam file overlapping the site and from those add up the coverage of every nucleotide in the splice site. This was done for all bam files to establish the distribution of the coverage in every sample. The output for every site was then written to a distinct tab delimited file with one column per sample and the rows corresponding to the position in the site. Now the coverage distribution was plotted with R.

The DESeq2 analysis was also executed with some variation, in order to assess the implication of the different steps. In particular the filter was skipped resulting in an analysis with every site regardless of its coverage. Also a lower filter with a minimum of 12 reads per clone or an average of 3 reads per sample was used. Finally the normalisation step was skipped, resulting in an analysis comparing the raw coverages of every splice site.

Tests

For most of the python code tests were written using the unittest module. If possible all cases that a function could encounter were addressed by these tests. These different cases were then tested in the helper functions, which take their arguments as integers or strings from the functions dealing with the bam file.

Results and discussion

The PCA plot from the differential gene expression analysis (fig 3a) shows, that on a gene level the wild type samples are clearly distinguishable from the knock out clones. But also the PCA plot from the results of the described differential splicing analysis shows a clear distinction of the wild type from the KO clones (fig. 3b). These findings indicate, that the depletion of FUS has a bigger effect on the transcriptome than any clone specific variation. This is the case for both the differential expression analysis and the described analysis for differential splicing. Hence the differences that are detected and described are not only caused by random effects, but are actually at least partially a true effect of the FUS depletion in our samples.

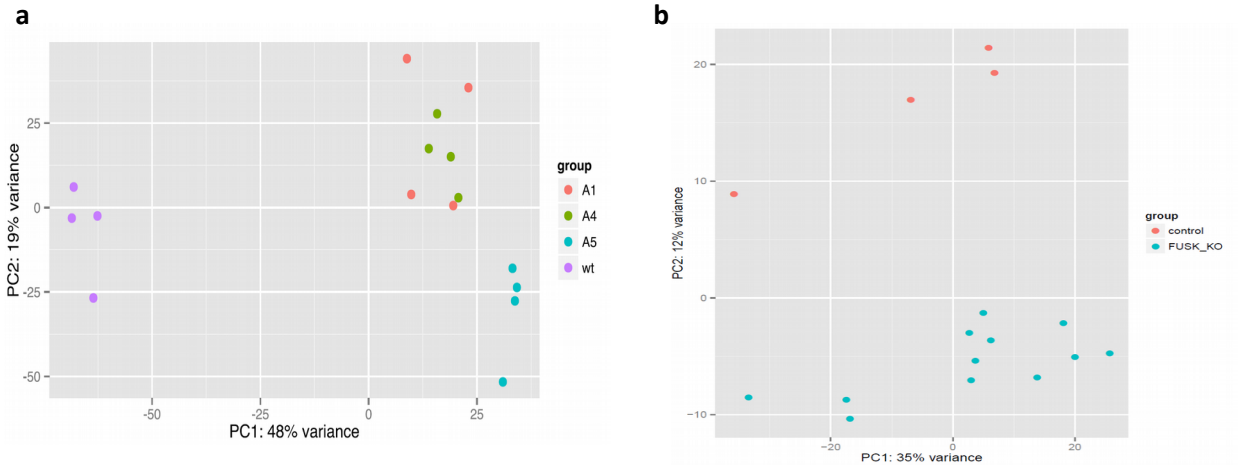

Figure 3 a) PCA plot of differential expression of the different clones, showing a clear distinction of wild type and KO clones. **b)** The PCA plot of differential splicing shows a clear separation of wild type and KO as well.

Even though every resulting site had to be checked for its properties, in order to assure their validity. This was only done for the ten most significant sites, in order to get a prove of principle and also to get an overview of the properties of the results, but it could easily be applied to any site of interest in an automatic manner.

The bar plots of the normalisation counts and the coverage revealed a large variation of the counts among the samples form one KO clone or wild type (fig 4). Some of the inspected sites even showed a critically large variation. Such sites have to be handled with care, it might be that part of their differential splicing is only noise. How ever for most of the sites the wild type showed a distinct pattern from all of the KO clones and the KO clones appear to be similar to each other. There fore it is probably beneficial to use all of the KO clones in the DESeq2 analysis, in order to handle the noisy data.

The plots showing the distribution of the coverage of the intron, can be used to check whether the site is a splice site, that does not contain a exon in an alternative splice product or any other abnormality that could interferer with the analysis. Only if the coverage is distributed evenly over the entire site. For most of the sites this was the case (fig. 5a). How ever the coverage of one site was concentrated on one half of the intron leaving the other half with almost no coverage (fig 5b). A likely explanation for this finding is that the site is subject to alternative splicing with an alternative splice site inside the intron, resulting in a splice variant with a longer exon including part of the detected intron. In that case it is possible that no read mapping across the alternative splice site with the shortened intron is present in the data set. This means that it is impossible to identify the site as an ambiguous site whose intron coverage can not be estimated. Therefore the reads considered to cover the intron might in fact originate from the alternatively elongated exon and not as assumed from the intron. This shows that the plot of the coverage distribution is a useful and also required tool for assuring the correctness of the identified sites.

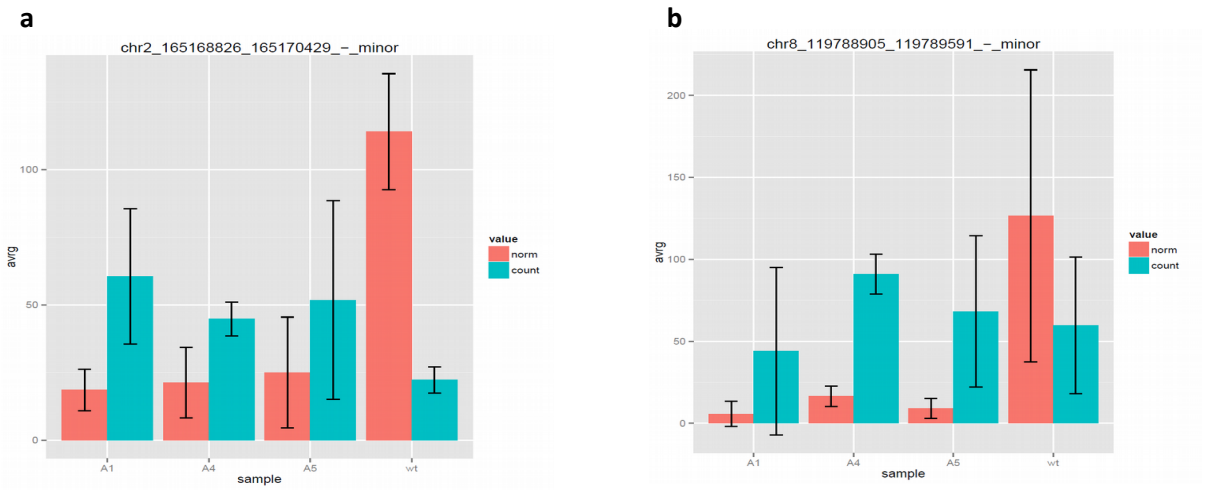

Figure 4: Bar plots for two of the ten mos significant sites showing the average counts, respectively normalisation counts with the standard deviation. Especially in **b)** the variation within the clones is strikingly large

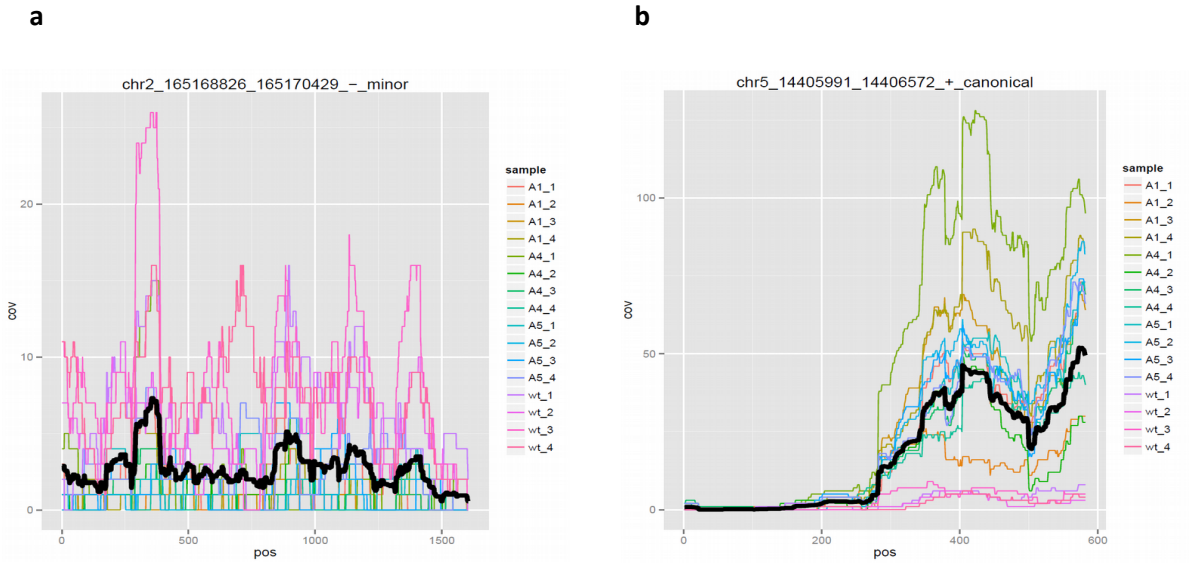

Figure 5: a) A plot of the coverage distribution showing an approximately equal distribution of the coverage over the entire site. **b)** A coverage plot for a site with an unequally distributed coverage.

The implication of the threshold of reads indicating alternative splicing in the estimation of the coverage of the intron was assessed by running the analysis with an alternative threshold of 0.05. This means that the coverage for sites with a ratio of reads indicating alternative splicing to normal reads smaller than 0.05 was estimated, and included in the analysis. Upon the statistical analysis it became recognizable that this threshold has no effect on the ten most significant results, they were exactly the same. Notably the PCA plot does not allow a distinction of wild type and KOs as well as in the standard analysis any more. Also the number of significant sites increased more than two fold on a significance level of 0.01. Hence when focusing on the most significant sites the standard threshold of zero seem reasonable. By that one can ensure that no alternatively spliced site interferes with with the results. However if a more comprehensive analysis is required a higher threshold can be considered. But in that case the evaluation of the resulting sites with the bar plot and the coverage distribution plot is even more important, because some of the results could be biased by alternative splicing.

The threshold for the coverage was also altered for the assessment of its implication. One analysis was run with a threshold of zero resulting in the inclusion of all sites, regardless of their coverage. This had a big effect on the result. First of all the PCA plot showed no reasonable distinction of wild type and KOs any more. Also the ten most significant results were different ones. The bar plots and the coverage distribution plot revealed that most of these sites have a intron coverage of zero or all most zero in both wild type and KOs. Thus the estimated change of splicing efficiency is only based on a different expression level, and are there fore not reliable. An other analysis with a threshold of 12 reads per clone respectively an average of three reads per sample per clone, showed similar results as the standard analysis with a threshold of 20 reads per clone. The ten most significant results were exactly the same and on a significance level of 0.01 only 36 more significant results could be found. Thus the threshold for the coverage is required for a reasonable analysis and a threshold of 20 reads per clone appears to be suitable.

Lastly the normalisation step was skipped resulting in an analysis which compares only the coverage counts, regardless of the normalisation counts. This variation too induced a break down of the PCA plot. Hence for a reliable analysis the normalisation with the number of reads mapping across the intron is required.

The exported result list from the standard analysis (0% ambiguous reads accepted, min. 20 reads per clone) was investigated for enrichment of minor splice sites among the sites with a significant p-value using bash. 44530 sites were considered in the analysis of which only 244 were annotated as minor splice sites (0.55%). However among the 244 sites with a p-value smaller than 0.01, 64 minor splice sites could be found, representing a fraction of 16.87% minor splice sites among the sites with a significantly altered splicing efficiency. Even more on a significance level of 0.005 313 sites were found to be significantly altered among which 64 were minor splice sites, representing a fraction of 20,45%. This distinctive enrichment of minor splice sites among the significant results supports the working hypothesis that FUS is involved in the splicing of minor splice sites.
Furthermore the function of the genes corresponding to the 15 most significant results were briefly investigated. This revealed two voltage dependent ion channels (VDAC3 and SCN3A) to be among the most significant results. This finding is particularly interesting, since such channels are crucial for the functioning of the neurons. This might be one of the pathways by which a loss of functional could FUS impair the functioning of neurons.

Conclusion and future perspectives

Many filters and thresholds have to be applied during the analysis. Without those the analysis is not reliable, because the unusable sites in the data set can not be excluded. Filters and thresholds are the only method to assure that alternative splicing and sites with low coverage do not interfere with the analysis. Though this implies that the analysis focuses on a small subset of the splice sites in the transcriptome. Not all sites can be respected in this analysis. Even though some interesting results such as the enrichment of minor splice sites among the significant results could be established.
Some improvement on the analysis could still be made. First of all it would be interesting to know if the differences between the KO clones has an effect on the results. Therefore a clone specific analysis could be performed. If the PCA plot with the differential splicing results shows a clear distinction of the KO clones, one would have to consider performing the analysis individually for every clone. If otherwise the variation within the clones is larger than the variation among the clones, the described standard analysis, which treats all of the clones as equal replicates, is probably the best approach.
Lastly the pipeline could be simplified. At the moment it consists of multiple python, bash and R files. Also it produces a lot of intermediate files. Thus it could be simplified by concatenating the files by making use of some interactions. This would make the pipeline much more user friendly, such that it could be used for further analysis in other research projects.

Acknowledgements
I thank Martino Colombo for providing and supervising this research project.

References

1. Zhou Y, Liu S, Liu G, Öztürk A, Hicks GG: ALS-Associated FUS Mutations Result in Compromised FUS Alternative Splicing and Autoregulation. PloS Genet 2013 9(10): e1003895. doi:10.1371/journal.pgen.1003895
2. Osamu Onodera, Tomohiko Ishihara, Atsushi Shiga, Yuko Ariizumi, Akio Yokoseki, Masatoyo Nishizawa: Minor splicing pathway is not minor any more: Implications for the pathogenesis of motor neuron diseases. Neuropathology 2014, 34, 99-107, doi:10.1111/neup.12070
3. http://samtools.github.io/hts-specs/SAMv1.pdf, 28.08.15
4. M. I. Love, W. Huber, S. Anders: Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2. Genome Biology 2014, 15:550

Apendix 1:
The python script used for processing the bam files.

```
import re
import os.path
import numpy as np
import subprocess as sp
import pysam as ps
import re

class Bam:
    """utility functions to parse bam"""

    def __init__(self, bam_path='', sam_data='', reads_orientation='forward'):
        """
        sam data can be provided for testing
        splices_dic contains all splice sites (key) mapped to a list with the count of the site and the intron
coverage
        :param reads_orientation: either 'forward' or 'reverse'
        """
        self.reads_orientation = reads_orientation
        self.splices_dic = {}
        if bam_path:
            assert bam_path[-4:] == '.bam'
            assert os.path.isfile(bam_path)
        elif sam_data:
            bam_path = self.Sam(sam_data = sam_data).toIndexedBam()
            os.remove('tmp.sam')
        else:
            return
        if not os.path.isfile(bam_path + '.bai'):
            pIndex = sp.Popen(['samtools', 'index', bam_path])
            pIndex.communicate()
```

```
        self.bam_file = ps.AlignmentFile(bam_path,'rb')

    def delete(self):
        os.remove(self.bam_file.filename.decode())
        os.remove(self.bam_file.filename.decode() + '.bai')

    def get_coverage(self, chrom, start, stop, min_qual=40):
        """
        get the number of reads in region
        reads is counted even if only 1 base overlaps region
        :param chrom: str chromosome name
        :param start: int start
        :param stop: int stop
        :param min_qual: default TopHat: only uniquely mapped reads
        """
        fetch = self.bam_file.fetch(chrom, start, stop)
        n_reads = 0
        for read in fetch:
            if read.mapq >= min_qual:
                n_reads += 1
        return n_reads

    def get_splicing_coverage(self, min_qual = 40, max_mm=0, max_sites_within = 0, min_overlap = 10):
        """
        get the number of reads from within splice site
        :param min_qual: default TopHat: only uniquely mapped reads
        :param max_mm: max tolerated mismatches on read ( including indel
        :param: min_overlap number of nucleotides inside splice junctions
        """
        for splice_site in self.splices_dic.keys():
            (chrom, start, stop, strand) = splice_site.split('_')
            start = int(start)
            stop = int(stop)
            (count, na) = self._count_coverage(chrom, start, stop, strand, min_qual, max_mm, min_overlap)
            if na > max_sites_within:
                self.splices_dic[splice_site].append('Na')
            else:
                self.splices_dic[splice_site].append(count)

    def _count_coverage(self, chrom, start, stop, strand,  min_qual, max_mm, min_overlap):
        """
        :param chrom: str chromosome name
        :param start: int start
        :param stop: int stop
        returns the number of reads from within given region
        """
        fetch = self.bam_file.fetch(chrom, start, stop)
        n_reads = 0
        n_na = 0
        for read in fetch:
            if read.mapq  >= min_qual and strand == self._determine_strand(read) and not
self._is_around_site(read, start, stop):
```

```
            count = self._read_in_intron(read, start, stop, max_mm, min_overlap)
            if  count <= 1:
                n_reads += count
            else:
                n_na += 1
        return (n_reads, n_na)

    def _read_in_intron(self, read, start, stop, max_mm, min_overlap):
        """
        checks if a read is fully inside region and a good enough match
        :param read: read of interest, pysam object
        :param start: satrt of region
        :param stop: end of region
        :param max_mm: tolerate number of mismatches
        :return: is_in_intron 0 not ,1 is in site 2: Na, other splice site within site thus biased coverage
        """
        is_in_intron = 1
        leng = 0
        mm = 0
        has_intron = False

        for c_part in read.cigar:
            if c_part[0] != 1: # insertion = 1 no extension on refference
                leng += c_part[1]
            if c_part[0] != 0:
                mm += c_part[1]
            if c_part[0] == 3:
                has_intron = True
        if start > read.reference_start: # read befor start
            overlap = leng - (start - read.reference_start)
        elif stop < read.reference_start + leng : # read longer than junction
            overlap = stop -read.reference_start
        else:
            overlap = leng
        if overlap < min_overlap or mm > max_mm:
            is_in_intron = 0
        if has_intron:
            is_in_intron = 2
        return is_in_intron

    def _is_around_site(self, read, start, stop):
        read_splicer = self._read_splicer(read.cigar, read.reference_start)
        read_splice_sites = read_splicer.get_sites()
        for site in read_splice_sites:
            if site[0] <= start and site[1] >= stop:
                return True
        else:
            return False

    def get_splice_sites(self, min_qual=40):
        """
        get splice sites counts as dictionary
```

```
    :param min_qual: default TopHat only uniquely mapped reads
    :return dict: {"chr1_12038_12759_+": 56, ...}
    """
    for read in self.bam_file.fetch():
        if read.mapq < min_qual:
            continue
        read_splicer = self._read_splicer(read.cigar, read.reference_start)
        read_splice_sites = read_splicer.get_sites()
        if not read_splice_sites:
            continue
        chrom = self.bam_file.getrname(read.reference_id)
        for read_splice_site in read_splice_sites:
            strand = self._determine_strand(read)
            self._add2dict(read_splice_site, chrom, strand)
    return self.splices_dic


def _add2dict(self, splice_site, chrom, strand):
    locus_string = '_'.join([str(x) for x in [chrom, splice_site[0], splice_site[1], strand]])
    if locus_string in self.splices_dic:
        self.splices_dic[locus_string][0] += 1
    else:
        self.splices_dic[locus_string] = [1]

def _determine_strand(self, read):
    strand_bool = True
    if read.is_reverse:
        strand_bool = not strand_bool
    if self.reads_orientation == 'reverse':
        strand_bool = not strand_bool
    if read.is_read2:
        strand_bool = not strand_bool
    return '+' if strand_bool else '-'


def get_cov_list(self, site, one_based = True):
    """
    :param site: str of the intronto be analyzed
    :param onebased: boolean set to true if site is indexed in one base
    :return:list of coverage per nucleatide of intron
    """
    (chrom, start, stop, strand, mode) = site.split('_')
    start = int(start)
    stop = int(stop)
    chrom = re.findall('[\d+XY]', chrom)[0]

    if one_based:
        start -=1

    leng = stop - start  # the index is 1 based thus include sturt:::

    cov_list = [0] * leng
```

```
        reads = self.bam_file.fetch(chrom, start, stop) #fetch uses 0 based indexing
        count = 0
        for read in reads:
            if self._determine_strand(read) == strand and not self._is_around_site(read, start, stop):
                cov_list = self._add_read(read, start, stop, cov_list)
        return(cov_list)


    def _add_read(self,read, start, stop, cov_list):

        read_len = 0
        for c_part in read.cigar:
            if c_part[0] == 0:
                read_len = read_len + c_part[1]

        rel_read_start = read.reference_start - start

        for pos in range(max(0, rel_read_start ), min(rel_read_start + read_len, stop - start)):
            cov_list[pos] += 1


        return(cov_list)
```

Appendix 2:
the bash script used for the concatenation of the resulting files of the previous steps in python. This was file writes the 16 files with the coverage count into on single file. The same was done with the normalisation counts with a corresponding bash script

```
#!/bin/bash

for file in count_sorted_*; do
cut -f1 $file >> tmp;
done

sort -u tmp >> count.txt

cols=""
count=1

for file in count_sorted*;do

cp count.txt count_tmp

join -a1 -o 0 $cols 2.2 -e 0 count_tmp <(sort -k1,1 $file) > count.txt;
let count=$count+1
cols="$cols 1.$count"
echo $cols
done
```