Long-Read Data Analysis from MinION Sequencing

Taylor Shishido, Ionis Pharmaceuticals

July 2019

Introduction

Due to the relative novelty of third-generation sequencing, a thorough analysis workflow for long-read sequencing data has not yet been developed. Here I present a computational pipeline optimized for analyzing long reads that integrates multiple open-source software.

This computational pipeline processes and analyzes long RNA reads generated by an ONT MinION flow cell. The starting input is raw current signals in FAST5 file format generated by the MinION software. It begins by basecalling these raw signal files generated from a run and produces fastq files of RNA reads. An analysis of run throughput and read quality is performed followed by file conversion and mapping of reads to a reference genome. Mapping quality is reported and analyzed. Mapped reads are corrected for noise and filtered. Unique isoforms are then defined and quantified by collapsing reads with similar exon-splice junction structures. The final isoforms are categorized by their productivity as full-length transcripts and annotated by gene. From that information, gene and pathway expression can be extrapolated. To explore structure, the isoforms can be visually analyzed on Integrative Genome Viewer.

This pipeline integrates Minimap2, Samtools, TranscriptClean, Flair, and Bedtools. It is compiled using Python, Command Line, and R code, and streamlined into multiple Jupyter Notebooks. Commands to install all necessary packages are included.

Within the tshishido/ folder, the pipeline consists of seven notebooks:

`Guppy.ipynb` ⇒ `Demultiplex.ipynb` ⇒ `Fasta.ipynb` ⇒ `Minimap2.ipynb` ⇒ `CoverageAnalysis.ipyng` ⇒ `Flair.ipynb` ⇒ `Enrichr.ipynb`

[width=]images/workflow.PNG

Figure 1: Novel pipeline transforms MinION output into quantified unique isoforms.

The "/tshishido/software/" folder contains both program and annotation/genome reference files, and the "/tshishido/X_notebooks/" folder contains discontinued work. All of the data gathered from previous sequencing runs is organized in "/tshishido/taylor_minion_data_analysis.xlsx".

Links to open source software:

I would also like to acknowledge Steven Kuntz, Sagar Damle, and Chris Hart for their guidance.

Minimap2:
https://github.com/lh3/minimap2
Samtools:
https://github.com/samtools/samtools
TranscriptClean:
https://github.com/dewyman/TranscriptClean
Flair:
https://github.com/BrooksLabUCSC/flair
Bedtools:
https://github.com/arq5x/bedtools

0.1 Basecalling

Basecalling by the ONT software Guppy was performed using batch computing through Amazon Web Services. Commands are given in `Guppy.ipynb`. A shell script starts an instance (8GB of memory capacity and 2 CPU cores with 2 threads per core) that installed and called Guppy for a single input file. This script was ran through a loop function for every FAST5 file of raw data generated by the sequencing run. Config files (found on the ONT website) specify the the kit, flowcell, and base-calling model parameters.

Example: for PCR-cDNA protocol and fast basecalling model, I used the config file dna_r9.4.1_450bps_fast.cfg. A sequence run that generated 144 FAST5 files with 50,000 reads each took around 2.5hrs to run.

0.2 Demultiplexing

`Demultiplex.ipynb` converts raw FASTQ output files to FASTA format and uses AWS parallelized computing and a multi-processor pooling function to demultiplex barcoded pcr-cDNA reads. The pooling function requires input in the format of a list of lists. Each nested list only contains a single FASTA-converted file. For a 24-mer barcode, 6 mismatches are tolerated. Only reads with barcodes ligated to the forward strand are considered. The FASTA files are merged together by sample and a breakdown of the yield by of reads for the run is given. `Demultiplex.ipynb` should be used to process all of the FASTQ files generated from a single run. Barcodes and primer sequences not removed for downstream analysis.

0.3 Analysis of Run Yield And Read Quality

`Fasta.ipynb` has two functions: 1) generate read length and overall throughput statistics for a FASTA file, and 2) visualize the read length distribution. There is a true-false parameter to specify making the histogram. This function should be performed for each sample (output FASTAs after demultiplexing). Running the function on the first and last FASTA files before multiplexing will give the consistency of data quality over time of the run.

0.4 Mapping

Minimap2 is a long-read mapper that utilizes reference genome indexing and splice-aware alignment. `Minimap2.ipynb` outputs commands that can be copied and pasted (using a Ctrl+Shift+v) into a Terminal window. Pre-built alignment parameters for different ONT read types (RNA, 1D cDNA) are specified in the arguments of `Minimap2.ipynb`'s main function. The reference genome, if a larger file, should be indexed into an .mmi file by Minimap2 (see cell below `Minimap2()`). Mapping one million reads to the human genome can take from 4 to 6 hours. Did not figure out a way to parallelize this function, although it might be useful.

NOTE: Mapping on an AWS instance would sometimes terminate early and produce a smaller truncated SAM file. Running Minimap2 on a local computer via Cygwin (paste using a middle click) doesn't have this issue. If transferring SAM files from local computer to an AWS machine, double-check file size.

The following reference genomes come from Ensembl.
File path: /scratch/tshishido/software/ref_fasta/

Genome	File name	Minimap2 Indexed FAST
human	homo_GRCh38_trimmed_ref.fa	homo_ref.mmi
mouse	mouse_GRCm38.p6_ref.fa	mouse_ref.mmi
Human chr.21	homo_chr21.GRCh38_trimmed_ref.fa	N/A

0.5 Coverage Analysis

There are three main functions in `CoverageAnalysis.ipynb`.

1. Samtools() The alignment SAM file (Minimap2 output) is converted to BAM format, indexed into an output file called "aln_index.csv," and sorted by chromosomal coordinates. Output reports a breakdown of QC-passed reads and the mapping rate. Reads that map to the main chromosomes are extracted into a new file for further analysis.

2. SplitSamTrCl() A new SAM file is generated from the subset of reads that align to each chromosome (labeled using RefSeq ids: NC_000001.11.sam, etc). The software TranscriptClean corrects mismatches and indels, which can be visualized in IGV, with a reference genome FASTA file (see table in mapping section). The demultiplex pooling function is used, though each nested list must now contain a chromosome id. An additional reference file of common human variants from dbSNP Build is used to avoid correcting away known splice variants.

TranscriptClean has four outputs per input file into a new folder "trcl_VA": a fasta and SAM file of the corrected reads and two full error log files. The SAM output files are aggregated using Samtools, and the other three types of output files are merged using bash commands. The aggregated SAM output is converted to BAM format ("trcl_VA/_cleansort.bam") and will be used for isoform identification and quantification with Flair.

A CSV of chromosome read coverage is generated by Samtools and stored in a directory titled 'chr_cvg' for generating color density plots with `R-plot.ipynb` if desired.

3. Error() Sequencing error rate is calculated by number of mismatches and indels detected, but not necessarily corrected, in the aggregated TranscriptClean log file. This calculation should be performed for at least one sample of a run, assuming read length distribution and quality was similar across all samples of the whole run.

NOTE 1. TranscriptClean is not run for reads mapped to the mouse

genome. The mismatch and indel noise is lighter than from human genome-mapped reads, and doesn't seem to affect the number of defined isoforms. The BAM file called "merged_mapped.bam" rather than "_cleansort.bam" is used for Flair.

NOTE 2. There is an additional TranscriptClean command (see Github wiki page) that generates an error correction report using R but is very memory- and CPU-intensive.

0.6 Coverage Visualization

`R-plot.ipynb` generates a coverage histogram for a chromosome based on its coverage file (CSV format) using a color density plot. Coverage values are binned by position to reduce computational cost, though the graph still functions to visually predict higher regions of coverage.

0.7 Isoform Identification and Quantification

`Flair.ipynb` uses three primary steps of the Flair pipeline: read correction by positionally adjusting nearly identical splice junctions to be identical, read collapsing into high-confidence isoforms, and quantifying these isoforms by aligning the un-mapped raw reads to them. The main function outputs commands to convert the BAM file of reads corrected by TranscriptClean to PSL format followed by the Flair primary steps (correct, collapse, quantify), that can be copied and pasted using a middle click into the Cygwin command window. The genome annotation files must be GTF format. The ones I used previously are downloaded from GENCODE. The PSL output file of splice-corrected reads ("flair_all_corrected.psl") and the raw un-mapped reads are used in the collapse function. This PSL file of collapsed unique isoforms ("flair.collapse.isoforms.psl") can be visualized with IGV, using the built-in hg38 human genome reference file. For mouse, I created my own genome file using igvtools. Primary full-length isoforms will have a transcript id consisting of "refseqtranscript_refseqgene," alternative full-length (or close to full-length) isoforms will be identified as "randomstring_refseqgene," and truncated transcripts will be labeled as "randomstring_chrcoordinate."

NOTE: There were multiple naming inconsistencies for the chromosome labels. The unchanged, inconsistent files are labeled "old.gencode..." Use

File path:
/scratch/tshishido/software/flair/

Genome	File name
human	new.gencode.v29.annotation.gtf
mouse	new.gencode.vM22.annotation.gtf

the cleaned annotation files labeled "new.gencode..." Corrections to use the "NC_0000" naming scheme were done manually.

Quantification is performed by counting the number of raw reads that align to a Flair-defined isoform, and the output files are CSV format recording the number count and TPM value for each isoform. For comparing expression of isoforms across samples or conditions, the FASTA outputs generated from Flair collapse should be merged before running Flair quantify.

0.8 Further Analysis

1. IsoformProductivity() This function uses an accessory Flair script. A gene is characterized as expressed if it has at least one multiple-exon isoform with a mature start and stop codon, or an isoform longer than 300bp containing at least one exon defined by Flair. Isoforms with a premature stop codon or shorter than 300bp are not considered to express a gene. The output is a breakdown of the isoform productivity and a PSL file "all_productive_isos" only containing gene-expressing isoforms.

2. IntronRetention() This function uses an accessory Flair script. Each isoform entry from the input PSL file is labeled as spliced or retaining and the function prints a breakdown of categorizing.

3. Bedtools() Isoforms are manually annotated by gene name using Bedtools. A compressed BAM file (sequences not included) generated by the `intersect` tool reports positional overlaps between a GTF-formatted file of isoforms and an genome annotation containing **ONLY** exons and genes (/tshishido/software/flair/exonsgenes.v29.annotation.gtf). From this output, the total number of isoforms is reported and a histogram of isoform counts per gene is displayed. A frequency table of isoforms for all expressed genes is created.

In Bedtools(), the output BAM file is read by each line. An entry that references a new transcript is added to a dictionary as a key with the gene

name as its value. If the transcript already appears in the primary dictionary with a different gene name, it is instead added to a list of transcripts with multiple annotations. This case addresses isoforms that intersected with multiple genes from the annotation file. From this list of already annotated transcripts, a separate dictionary is created where the transcript is the key and the value of each key is a list of genes parsed from every entry for that transcript in the annotation file. A gene frequency table for each transcript is generated, and the gene with the highest frequency becomes the final annotation for that gene. More notes on this in Areas of Improvement.

3. `Enricher.ipynb` This script has packages and commands to create two or three-circle Venn diagrams. It also creates a dataframe for Enrichr results that are saved to a CSV. There is also code that identifies genes expressed uniquely to a sample.

4. Using IGV You can zoom in on a specific gene region of a genome just by entering in a gene name or transcript id. Isoforms should always be viewed in Squished mode. IGV was used while quantifying expression since visual interpretation of what constitutes an isoform of sufficient length was needed.

0.9 Areas of Further Improvement

1. I'm not certain if my code to account for transcripts that overlapped multiple genes functions accurate. I was in the process of debugging it, and left some debug code at the bottom of the notebook `Flair.ipynb`.

2. The number of isoforms extrapolated from the bedtools intersect output BAM file is consistently less than the number of isoforms extrapolated from the PSL file (input file for `bedtools intersect`). Only about 80-90% of the isoforms are recovered.

3. Other accessory Flair scripts are listed in the "Other notes" in the last cell of `Flair.ipynb`. Some seemed useful, but I never got around to running them.