Stranded Mapping from Long Reads Version 3

David Eccles

Abstract

This protocol demonstrates how to convert raw long reads produced using a strand-specific sequencing protocol (e.g. ONT's strand-switching protocol) into strand-specific mapped reads.

The general ides is to use LAST to identify the adapter orientation relative to the genome, and then use that information to reorient called sequences to create a stranded BAM files that is displayable in a genome browser.

Citation: David Eccles Stranded Mapping from Long Reads. **protocols.io**
dx.doi.org/10.17504/protocols.io.syheft6
Published: 03 Sep 2018

Before start

I have written my own script to process LAST results into a CSV format, which makes it easier to do line-by-line data filtering. I have also created a fastq filtering script that helps for filtering reads into different files.

You will also need access to the following free and open-source software programs:

- LAST
- minimap2
- samtools

And the following additional data files:

- a FASTA file containing strand-specific primer / adapter sequences.
- a FASTA file containing the genome / sequence of interest.

Protocol

Index Preparation

Step 1.

Prepare genome index for spliced alignment

SOFTWARE PACKAGE (Linux)

minimap2, 2.10

Heng Li
https://github.com/lh3/minimap2

cmd COMMAND

```
minimap2 -d mmus_ucsc_all-splice.idx -Q -t 10 -x splice mmus_ucsc_all.fa
```

Index Preparation

Step 2.

Prepare barcode and inner primer adapter indexes

SOFTWARE PACKAGE (Debian GNU/Linux)

LAST

Martin Frith
http://last.cbrc.jp/last/

cmd COMMAND

```
lastdb -uNEAR -R01 PCR_barcode_base.fa PCR_barcode_base.fa
lastdb -uNEAR -R01 adapter_seqs.fa adapter_seqs.fa
```

Index the adapter files (used for demultiplexing)

Read Correction

Step 3.

Download canu v1.7.1 or later

SOFTWARE PACKAGE (Linux / Darwin)

canu, 1.7.1

Maryland Bioinformatics Labs
https://github.com/marbl/canu

Read Correction

Step 4.

Collate basecalled reads (yes, all of them, and ignore the barcode assignments).

cmd COMMAND

```
pv workspace/fail/*/*.fastq | gzip > called_fail.fastq.gz
pv workspace/pass/*/*.fastq | gzip > called_pass.fastq.gz
```

Collate basecalled reads into separate files for pass and fail (but all barcodes thrown together)

Read Correction

Step 5.

I prefer starting off my data analysis with a read correction with Canu. I use `minimap` as the mapper to speed this up. The `genomeSize` parameter should be approximately a tenth to a fortieth of the number of bases in your dataset to make sure that no sequences are excluded (bigger is better, as long as Canu doesn't freak out about memory consumption):

(This creates a file `canu_corrected/canu_corrected.correctedReads.fasta.gz`)

cmd COMMAND
```
canu -correct overlapper=minimap genomeSize=400M \
  minReadLength=100 minOverlapLength=30 -p canu_corrected -d canu_corrected -nanopore-
raw ./called_pass.fastq.gz \
  ./called_fail.fastq.gz
```
Correct Reads with canu (both passed and failed sequences), using minimap as the mapper

Read Correction

Step 6.

Identify corrected reads using fastx-length.pl

cmd COMMAND
```
pv canu_corrected/canu_corrected.correctedReads.fasta.gz | \
  fastx-length.pl | awk '{print $2}' | gzip > names_corrected_all.txt.gz
```
create list of corrected sequence lengths

Read Correction

Step 7.

Extract uncorrected reads using fastx-fetch.pl

cmd COMMAND
```
pv called_pass.fastq.gz called_fail.fastq.gz | \
  fastx-fetch.pl -v -i names_corrected_all.txt.gz | gzip > uncorrected_all.fastq.gz
```
filter/extract uncorrected reads

Read Correction

Step 8.

Join corrected and uncorrected reads. The uncorrected reads are converted to fasta format with fastq2fasta.pl to make the joined file formats consistent.

cmd COMMAND
```
pv uncorrected_all.fastq.gz | zcat | fastq2fasta.pl | gzip > uncorrected_all.fasta.gz
pv uncorrected_all.fasta.gz canu_corrected/canu_corrected.correctedReads.fasta.gz | zcat |
\
  gzip > uncorrected_corrected_all.fasta.gz
```
Concatenate corrected reads to uncorrected reads

Demultiplexing

Step 9.

The next step I carry out is a basic read-level QC to exclude chimeric reads. Porechop can be used for this, although that removes adapters by default, which is not particularly useful in this case.

I use LAST to search for adapter sequences within the corrected reads, pass it through my conversion script, and extract out duplicated mappings (i.e. where the same read/adapter pair appears more than once in the mapping results).

The first phase of this is mapping to barcode sequences to generate a CSV file of assignments. The corrected and uncorrected reads are mapped separately to give the uncorrected reads the best chance of mapping with '-Q 1'; the corrected reads are in FASTA format, so the corrected mapping does not use quality scores.

Note that this isn't a perfect mapping. Due to the lack of adapter sequences, these barcode sequences alone can match sequences that are inside the sequence of interest. On the plus side, there won't be any spurious matches to adapters from other barcode sequences if the correct barcode has too many errors.

cmd COMMAND

```
(lastal -
P 10 barcode_base.fa <(pv canu_corrected/canu_corrected.correctedReads.fasta.gz | zcat);
lastal -P 10 -Q 1 barcode_base.fa <(pv uncorrected_all.fastq.gz | zcat)) | \
  maf_bcsplit.pl | gzip > barcode_assignments_all.csv.gz
```

Map to barcode sequences (excluding adapters)

Demultiplexing

Step 10.

Map to inner cDNA / adapter sequences to generate CSV file of assignments. The corrected and uncorrected reads are mapped separately because the corrected reads are in FASTA format.

cmd COMMAND

```
(lastal -
P 10 adapter_seqs.fa <(pv canu_corrected/canu_corrected.correctedReads.fasta.gz | zcat);
lastal -P 10 -Q 1 adapter_seqs.fa <(pv uncorrected_all.fastq.gz | zcat)) | \
  maf_bcsplit.pl | gzip > adapter_assignments_all.csv.gz
```

Map to cDNA adapter sequences

Demultiplexing

Step 11.

Chimeric reads (containing adapter sequences from different barcodes) are excluded, and the inner adapters are tallied by creating 'wide' tables indicating barcode/adapter assignments. This R script creates files 'barcode-adapter_assignments_ideal.csv.gz' and 'barcode-adapter_assignments_valid.csv.gz'.

```
#!/usr/bin/env Rscript
bc.df <- read.csv("barcode_assignments_all.csv.gz");
ad.df <- read.csv("adapter_assignments_all.csv.gz");

library(dplyr);
library(tidyr);

## Create table of adapter additions
ad.tbl <- group_by(ad.df, query, target, dir) %>% summarise() %>%
    unite(tdir, target, dir, sep=".") %>% mutate(present=TRUE) %>%
    spread(tdir, present);

## collapse multiple query/target pairs into one
bc.tbl <- group_by(bc.df, query, target) %>% summarise(dir=paste(unique(dir),
collapse="/"));
bc.wide <- spread(bc.tbl, target, dir);

## identify reads with a unique barcode
bc.unique.tbl <- group_by(bc.tbl, query) %>% summarise(n = n()) %>%
    filter(n == 1) %>% select(-n) %>% left_join(bc.tbl, by="query") %>%
    left_join(ad.tbl, by="query", copy=TRUE);

bc.unique.tbl$`ONT_SSP.-`[is.na(bc.unique.tbl$`ONT_SSP.-`)] <- FALSE;
bc.unique.tbl$`ONT_SSP.+`[is.na(bc.unique.tbl$`ONT_SSP.+`)] <- FALSE;
bc.unique.tbl$`ONT_VNP.-`[is.na(bc.unique.tbl$`ONT_VNP.-`)] <- FALSE;
bc.unique.tbl$`ONT_VNP.+`[is.na(bc.unique.tbl$`ONT_VNP.+`)] <- FALSE;

colnames(bc.unique.tbl) <-
c("query","target","bcDir","SSPrev","SSPfwd","VNPrev","VNPfwd");

## read is considerd "valid" (for now) if at least one primer matches
bc.valid.tbl <- filter(bc.unique.tbl, (SSPrev | VNPfwd | VNPrev | SSPfwd));
## ideal reads have forward and reverse cDNA adapters in opposing
orientations
bc.ideal.tbl <- filter(bc.unique.tbl, ((SSPrev & !SSPfwd & VNPfwd & !VNPrev)
| (!SSPrev & SSPfwd & !VNPfwd & VNPrev)));

write.csv(bc.ideal.tbl, row.names=FALSE, file=gzfile("barcode-
adapter_assignments_ideal.csv.gz"), quote=FALSE);
write.csv(bc.valid.tbl, row.names=FALSE, file=gzfile("barcode-
adapter_assignments_valid.csv.gz"), quote=FALSE);
```

Demultiplexing

Step 12.

Create a list of used barcodes

cmd COMMAND

```
zcat barcode-adapter_assignments_ideal.csv.gz | tail -n +2 | awk -
F',' '{print $2}' | sort | uniq > used_barcodes.txt
```

Create a list of used barcodes

Demultiplexing

Step 13.

Demultiplex valid reads by barcodes using fastx-fetch.pl

cmd COMMAND

```
cat used_barcodes.txt | while read bc
  do echo "** ${bc} **"
  mkdir -p demultiplexed/${bc};
  pv uncorrected_corrected_all.fasta.gz |  \
    ~/scripts/fastx-fetch.pl -i <(zgrep ${bc} barcode-
adapter_assignments_ideal.csv.gz | awk -F',' '{print $1}')  | \
  gzip > demultiplexed/${bc}/${bc}_reads_all.fasta.gz;
done
```

Demultiplex valid reads by barcode

Demultiplexing

Step 14.

Demultiplex barcode-demultiplexed reads by SSP direction.

Note that the last four values in the 'wide' table refer to the reverse and forward mappings of the SSP and VNP primers respectively). The reverse reads are reverse-complemented with fastx-rc.pl, followed by a final concatenation to simplify the subsequent alignment steps.

cmd COMMAND

```
cat used_barcodes.txt | while read bc
  do echo "** ${bc}/fwd **";
  pv demultiplexed/${bc}/${bc}_reads_all.fasta.gz |  \
    ~/scripts/fastx-fetch.pl -i <(zgrep 'FALSE,TRUE,TRUE,FALSE$' barcode-
adapter_assignments_ideal.csv.gz | awk -F',' '{print $1}')  | \
  gzip > demultiplexed/${bc}/${bc}_reads_fwd.fasta.gz;
  echo "** ${bc}/rev **";
  pv demultiplexed/${bc}/${bc}_reads_all.fasta.gz |  \
    ~/scripts/fastx-fetch.pl -i <(zgrep 'TRUE,FALSE,FALSE,TRUE$' barcode-
adapter_assignments_ideal.csv.gz | awk -F',' '{print $1}')  | \
  fastx-rc.pl | gzip > demultiplexed/${bc}/${bc}_reads_rev.fasta.gz;
  pv demultiplexed/${bc}/${bc}_reads_fwd.fasta.gz demultiplexed/${bc}/${bc}_reads_rev.fasta
.gz | zcat | \
    gzip > demultiplexed/${bc}/${bc}_reads_dirAdjusted.fasta.gz
done
```

demultiplex demultiplexed reads by direction

Read Mapping

Step 15.

Download samtools v1.8 or later

SOFTWARE PACKAGE (Linux)
SAMtools, 1.8
Wellcome Trust Sanger Institute
https://github.com/samtools/samtools

Read Mapping

Step 16.

Now that the reads have been demultiplexed and oriented, the mapping can be done. I use `minimap2` for mapping long reads to a transcriptome. This creates '.bam' files in the 'demultiplexed' directory.

This is where the reverse complementing done during demultiplexing gives a big saving of effort.

cmd COMMAND
```
cd demultiplexed;
for x in BC*;
  do echo ${x};
  ~/install/minimap2/minimap2 -Q -t 10 -a -x splice mmus_ucsc_all-
split.idx ${x}/${x}_reads_dirAdjusted.fasta.gz | \
    samtools view -b | samtools sort > mm2_called_all_${x}_vs_MmusG.bam;
done
```
Mapping reads to an indexed transcriptome using minimap2.

Creating BigWig Coverage Files

Step 17.

Within the same 'demultiplexed' directory, a bedGraph of coverage is created using samtools mpileup and mpileupDC.pl, excluding any skipped intronic sequence. When 'mpileupDC.pl' is provided with a single file, it will output a bedGraph file with a header line starting with '##'; this header line is removed. The particular JBrowse plugin that I use for stranded display requires that the reverse strand have *negative* coverage values, so that file needs to be changed:

cmd COMMAND
```
for x in BC*;
  do echo ${x};
  samtools view -b -F 0x10 mm2_called_all_${x}_vs_MmusG.bam | \
    samtools mpileup -A -B -Q 0 -q 0 -I -q 0 -Q 0 - | \
    mpileupDC.pl | tail -n +2 > mm2_called_all_${x}_vs_MmusG.bg.plus
  samtools view -b -f 0x10 mm2_called_all_${x}_vs_MmusG.bam | \
    samtools mpileup -A -B -Q 0 -q 0 -I -q 0 -Q 0 - | \
    mpileupDC.pl | tail -n +2 > mm2_called_all_${x}_vs_MmusG.bg.minus
  perl -i -pe 's/([0-9]+)$/-$1/' mm2_called_all_${x}_vs_MmusG.bg.minus
done;
```
Generate bedGraphs from both forward and reverse reads

Creating BigWig Coverage Files

Step 18.

Stranded bedgraph files are converted to bigwig. This requires BEDTools and a genome information

file containing chromosome lengths (one for Mmus/mm10 is attached to this step).

SOFTWARE PACKAGE

BEDTools, 2.26.0

Quinlan laboratory, University of Utah
https://github.com/arq5x/bedtools2/

cmd COMMAND

```
for x in BC*
  do echo ${x}
  basename="mm2_called_${x}_all_vs_mmusAll"
  bedGraphToBigWig ${basename}.bg.plus Mmus_genome.chrInfo.txt ${basename}.bw.plus
  bedGraphToBigWig ${basename}.bg.minus Mmus_genome.chrInfo.txt ${basename}.bw.minus
done
```

Convert bedgraph to bigwig

JBrowse Configuration

Step 19.

Each track should have its own JBrowse configuration section using the *StrangedBigWig* class and *StrandedXYPlot* type. An example is shown here:

```
[tracks.BWCG004-4T1-BC04-both-track ]
storeClass     = StrandedPlotPlugin/Store/SeqFeature/StrandedBigWig
urlTemplate    = bw/mm2_called_CG004_BC04_vs_MmusG.bw
category       = MinION - Coverage
type           = StrandedPlotPlugin/View/Track/Wiggle/StrandedXYPlot
key            = MinION minimap2 coverage from CG004-4T1-WT (combined
strands)
scale          = log
scoreType      = maxScore
autoscale      = global
style.pos_color = darkred
style.neg_color = darkgreen
```

Sanity Check

Step 20.

If this has worked properly, then mapping human or mouse to the mitochondrial genome should show most expression appearing on the positive strand, with a small scattering of negative-strand expression, a bit like the *Expected Results* shown here.

If not, check for the following issues:

- Tracks not displaying at all in JBrowse -- make sure track IDs inside square brackets are of the form [*tracks.<unique-id-without-dots>-track*]
- JBrowse track is reflected in the X axis -- make sure that the reverse bedgraph file is orientated the correct way; it should be created with the '-f 0x10' flag (noe capitalisation.

- JBrowse track only shows one direction -- make sure that the reverse bedgraph file has *negative* values, and re-generate the bigwig file

EXPECTED RESULTS