Oct 10, 2019

Preparing Reads for Stranded Mapping V.6

David A. Eccles[1]

[1]Malaghan Institute of Medical Research (NZ)

In Development dx.doi.org/10.17504/protocols.io.74vhqw6

David A. Eccles
Malaghan Institute of Medical Research (NZ)

ABSTRACT

This protocol is for preparing long reads for stranded mapping, as an intermediate step for additional protocols:

- Aligning strand-oriented sequences to a transcriptome for transcript / gene counting
- Aligning strand-oriented sequences to a genome for confirmatory QC

Input(s): demultiplexed fastq files (see protocol Demultiplexing Nanopore reads with LAST), adapter file (containing strand-sensitive adapter sequences)

Output(s): oriented read files, as gzipped fastq files

Barcode Demultiplexing

1 Demultiplex reads as per protocol Demultiplexing Nanopore reads with LAST.

If this has been done, then the following command should produce output without errors:

```
for bc in $(awk '{print $2}' barcode_counts.txt);
  do ls demultiplexed/reads_${bc}.fq.gz;
done
```

Example output:

```
demultiplexed/reads_BC03.fq.gz
demultiplexed/reads_BC04.fq.gz
demultiplexed/reads_BC05.fq.gz
demultiplexed/reads_BC06.fq.gz
demultiplexed/reads_BC07.fq.gz
demultiplexed/reads_BC08.fq.gz
```

If the *barcode_counts.txt* file is missing, the output will look like this:

```
awk: fatal: cannot open file `barcode_counts.txt' for reading (No such file or directory)
```

If one or more of the barcode-demultiplexed files are missing, the output will look something like this:

```
demultiplexed/reads_BC03.fq.gz
demultiplexed/reads_BC04.fq.gz
demultiplexed/reads_BC05.fq.gz
ls: cannot access 'demultiplexed/reads_BC06.fq.gz': No such file or directory
ls: cannot access 'demultiplexed/reads_BC07.fq.gz': No such file or directory
demultiplexed/reads_BC08.fq.gz
```

Index Preparation

2 Prepare a FASTA file containing adapter sequences (see attached FASTA file).

adapter_seqs.fa

3 Prepare a substitution matrix for barcode mapping. The default substitution matrix is swayed too much by INDELs in the barcode sequences, so here's one that I've developed using a combination of trial & error and last-train:

```
#last -Q 0
#last -a 10
#last -A 10
#last -b 5
#last -B 5
#last -S 1
# score matrix (query letters = columns, reference letters = rows):
       A      C      G      T
A      4    -24     -9    -24
C    -24      5    -24    -14
G     -9    -24      7    -24
T    -24    -14    -24      8
```

bc.mat

[note: this is the same matrix as used for demultiplexing]

4 Prepare the LAST index for the adapter file. Following Martin Frith's recommendation, the '-uNEAR' seeding scheme is used to slightly increase sensitivity. This will generate seven additional files of the form <index name>.XXX:

```
lastdb -uNEAR adapter_seqs.fa adapter_seqs.fa
```

Orienting Reads

5 Map the reads to the adapter sequences using the previously defined substitution matrix. In this case it's important that the direction of mapping is also recorded, so the *cut* command selects three fields (query name [7], target name [2], mapping direction [10]):

```
for bc in $(awk '{print $2}' barcode_counts.txt);
  do echo "** ${bc} **";
  lastal -Q 1 -P10 -p bc.mat adapter_seqs.fa <(pv demultiplexed/reads_${bc}.fq.gz) | \
    maf-convert -n tab | cut -f 2,7,10 | uniq | \
    gzip > demultiplexed/adapter_assignments_${bc}.txt.gz
done
```

6 The adapter assignments are filtered through *uniq* in order to catch (and exclude) any reads with the strand-switch primer matching multiple times. To unpack the *uniq* pipe a little bit more, it skips the first field (adapter name), then matches up to 36 characters, retaining only lines that don't match any others. This catches a few more chimeric reads that were missed by the unique barcode filter in the previous protocol.

Reads are filtered into two groups (and one group-by-omission) based on the mapped direction of the strand-switch primer, then reverse-complemented (if necessary) to match the orientation of the original RNA strand. I use my fastx-fetch.pl and fastx-rc.pl scripts for this.

fastx-fetch.pl

fastx-rc.pl

```
mkdir -p oriented
for bc in $(awk '{print $2}' barcode_counts.txt);
 do echo "** ${bc} **";
 fastx-fetch.pl -i <(zgrep '^SSP' demultiplexed/adapter_assignments_${bc}.txt.gz | \
     sort | uniq -f 1 -w 36 -u | \
     awk '{if($3 == "+"){print $2}}') <(pv demultiplexed/reads_${bc}.fq.gz) | \
   gzip > oriented/${bc}_reads_fwd.fq.gz
 fastx-fetch.pl -i <(zgrep '^SSP' demultiplexed/adapter_assignments_${bc}.txt.gz | \
     sort | uniq -f 1 -w 36 -u | \
     awk '{if($3 == "-"){print $2}}') <(pv demultiplexed/reads_${bc}.fq.gz) | \
   fastx-rc.pl | gzip > oriented/${bc}_reads_rev.fq.gz
done
```

7 Forward and reverse-oriented sequences are combined together to form a single group of RNA-oriented reads.

```
for bc in $(awk '{print $2}' barcode_counts.txt);
  do echo "** ${bc} **";
  pv oriented/${bc}_reads_fwd.fq.gz oriented/${bc}_reads_rev.fq.gz | \
    zcat | gzip > oriented/${bc}_reads_dirAdjusted.fq.gz
done
```

Downstream Workflows

8 Following on from here, the oriented reads can be mapped to a genome (e.g. for visual confirmation of mapping), or to a transcriptome (e.g. for read counting):

- Stranded Mapping from Oriented Long Reads
- Stranded Transcript Count Table Generation from Long Reads