VERSION 8
NOV 07, 2022

SHARE

WORKS FOR ME 1

Plasmid Sequence Analysis from Long Reads V.8

DOI
dx.doi.org/10.17504/protocols.io.36wgq4n5yvk5/v8

David A Eccles[1]

[1]Malaghan Institute of Medical Research (NZ)

High molecular weight DNA extraction from all kingdoms
Tech. support email: **See@each.protocol**

David A Eccles
Malaghan Institute of Medical Research (NZ)

COMMENTS 0

ABSTRACT

This protocol demonstrates how to assemble reads from plasmid DNA, and generate a circularised and non-repetitive consensus sequence

At the moment, this protocol uses Canu to de-novo assemble high-quality single-cut reads.

Input(s):

- demultiplexed fastq files (see protocol Demultiplexing Nanopore reads with LAST). I've noticed that the default demultiplexing carried out by Guppy (at least up to v4.2.2, as used in the first version of this protocol) has issues with chimeric reads, which can affect assembly.

Output(s):

- Consensus sequence per barcode as a fasta file

DOI

dx.doi.org/10.17504/protocols.io.36wgq4n5yvk5/v8

PROTOCOL CITATION

David A Eccles 2022. Plasmid Sequence Analysis from Long Reads. **protocols.io**
https://dx.doi.org/10.17504/protocols.io.36wgq4n5yvk5/v8
Version created by David A Eccles

CREATED

Nov 07, 2022

LAST MODIFIED

Nov 07, 2022

PROTOCOL INTEGER ID

72362

BEFORE STARTING

When preparing plasmid DNA for sequencing, I follow one piece of advice that Dr. Divya Mirrington (ONT) gave me about pooling: create a pooled sample with volumes that you're confident about, then remove an aliquot from that for adding the remaining reagents [paraphrased]. I don't do any additional cleanup for purified plasmid DNA; they tend to sequence very well on flow cells without that cleanup.

My key requirement for plasmid sequencing is a concentration of >20 ng/µl (ideally by qubit or quantus). Concentrations over 100 ng/µl should be diluted down. If the plasmids can be diluted down to all exactly the same concentration (but at least 20 ng/µl), or they're all similar lengths, then that makes creating an equimolar pool much easier.

When creating the pools, I add at least 1 µl of the the sample that I need to add the least for (might be more if the total volume is less than 11 µl), then add the corresponding amount of other samples to create an equimolar pool. I then take 11 µl from the pool to be used for rapid adapter addition.

If samples are equal concentration:
Add amounts according to the length of the plasmid divided by the length of the shortest plasmid. For example, if there are two plasmids, one with length 3kb and another with length 35 kb, then add 1 µl of the 3kb plasmid, and 35/3 = 11.7 µl of the 35kb plasmid.

If plasmids are roughly equal length (i.e. less than ~10% length difference between plasmids):
Add amounts according to the concentration of the highest-concentration sample divided by the concentration of the plasmid. For example, if there are three plasmids, one with concentration 50 ng/µl, one with concentration 35 ng/µl, and one with concentration 20 ng/µl, then add 1 µl of the 50 ng/µl plasmid, 50/35 = 1.4 µl of the 35 ng/µl plasmid, and 2.5 µl of the 20 ng/µl plasmid. The total volume of this pool will be less than 11 µl (1 + 1.4 + 2.5 = 4.9 µl), so in this case I would triple these volumes (3 µl; 4.2 µl; 7.5 µl) to create a pool of > 11 µl.

If samples are different concentrations and different lengths:

Make the sample prep easier. Use multiple flow cells for different plasmid length ranges. Dilute higher-concentration samples down to the lowest-concentration samples. I don't recommend trying to do both calculations at the same time to determine added volumes because there's a much higher chance of getting added amounts wrong, leading to wasted samples or wasted flow cells.

If you have a sufficiently-accurate pipetting robot, a sample sheet, and someone who is comfortable with equations:
Pre-calculate amount to add assuming 12 µl total pool volume:
ratio = *length* / max(*length*) * max(*conc*) / *conc*
volume = *ratio* * 12 / sum(*ratio*)

[That's my guess at the right equations; please let me know if there's an error]

Read file preparation

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_BC02.fq.gz
demultiplexed/reads_BC03.fq.gz
demultiplexed/reads_BC04.fq.gz
demultiplexed/reads_BC05.fq.gz
demultiplexed/reads_BC07.fq.gz
demultiplexed/reads_BC09.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_BC02.fq.gz
demultiplexed/reads_BC03.fq.gz
demultiplexed/reads_BC04.fq.gz
ls: cannot access 'demultiplexed/reads_BC05.fq.gz': No such file or
directory
ls: cannot access 'demultiplexed/reads_BC07.fq.gz': No such file or
directory
demultiplexed/reads_BC09.fq.gz
```

1.1 If reads have been demultiplexed by MinKNOW, then the following approach should work to create the right input format:

```
mkdir demultiplexed;
# readlocation is the directory that contains barcode subdirectories
readLocation = "../*/fastq_pass"; # or "../*/pass" for Guppy
for x in $(ls ${readLocation}); do
  echo ${x};
  cat ${readLocation}/${x}/*.fastq | gzip >
demultiplexed/reads_${x}.fq.gz;
  echo "1 ${x}" >> barcode_counts.txt;
done
```

2 Create a directory to store results files

```
mkdir results
```

3 Determine the N50/L50 read length for each barcode. This will be used as the initial guess at the assembly size.

```
(for bc in $(awk '{print $2}' barcode_counts.txt);
    do echo -n ${bc};
    fastx-length.pl demultiplexed/reads_${bc}.fq.gz 2>&1 > /dev/null | \
      grep L50 | awk '{print "\t"$5$6}' | perl -pe 's/b$//';
 done) > results/read_L50.txt
```

This file can be viewed to confirm the assembly lengths:

```
cat results/read_L50.txt
```

Example output:

```
BC02    347
BC03    8.904k
BC04    8.888k
BC05    10.262k
BC07    11.076k
BC09    11.093k
```

Note: this file will be used for subsequent downstream processing. If any of these barcodes shouldn't be processed further, feel free to remove the corresponding lines from this file

Read filitering

4 Filter out any reads that are less than half of the target read length, and determine the average quality of the remainder, keeping information on (at most) 100 of the highest-quality reads:

```
cat results/read_L50.txt | while read bc len;
  do echo ${bc} ${len};
  ~/scripts/fastx-compstats.pl demultiplexed/reads_${bc}.fq.gz | \
    sort -t ',' -k 16rn,16 | \
    awk -F ',' -v "len=${len}" \
      'BEGIN{if(match(len, "k") > 0){sub("k","",len); len=len*1000}}
       {if($18 > (len / 2)){print}}' | \
    head -n 100 | grep -v '^name' > results/bestLong_100X_${bc}.csv;
done
```

Check how successful the sequencer was at getting 100X coverage by counting lines:

```
wc -l results/bestLong_100X_*.csv
```

Example output. In this case BC02 has fewer than 100 reads, so the assembly is less likely to be high-quality:

```
  90 results/bestLong_100X_BC02.csv
 100 results/bestLong_100X_BC03.csv
```

```
  100 results/bestLong_100X_BC04.csv
  100 results/bestLong_100X_BC05.csv
  100 results/bestLong_100X_BC07.csv
  100 results/bestLong_100X_BC09.csv
 590 total
```

5 Subset the original read sets to only include the high-quality long reads:

```
cat results/read_L50.txt | while read bc len;
  do echo ${bc} ${len};
  ~/scripts/fastx-fetch.pl -i results/bestLong_100X_${bc}.csv \
    demultiplexed/reads_${bc}.fq.gz > results/bestLong_100X_${bc}.fastq;
done
```

Choice 1: mapping to a reference sequence

6 This protocol assumes the reference sequence has a name 'refName', and can be found at 'reference/refName.fa'. Change 'refName' here in the likely event that the sequence / file has a different name, then export the variable *REFNAME* to make subsequent steps easier.

```
export REFNAME="refName"
```

7 Create a LAST index for the reference sequence:

```
lastdb -uNEAR -R01 reference/${REFNAME}.fa reference/${REFNAME}.fa
```

8 Prepare a substitution matrix for barcode mapping. I recommend doing this by training LAST, and incorporating quality scores into the trained matrix:

```
last-train --matsym -Q 1 -P 10 reference/${REFNAME}.fa \
  results/bestLong_100X_BC*.fastq
```

Copy the last few lines from the output into a text file called `*plasmid.mat*`

Here is a matrix that I have created from plasmid reads called using the guppy super-accuracy basecaller, v5.0.13:

```
#last -Q 1
#last -t4.39812
#last -a 17
#last -A 18
#last -b 4
#last -B 5
#last -S 1
# score matrix (query letters = columns, reference letters = rows):
       A      C      G      T
A      6    -45    -34    -46
C    -45      6    -46    -45
G    -34    -46      6    -46
T    -46    -45    -46      6
```

plasmid.mat

This matrix has a moderate penalty for opening gaps (i.e. insertions and deletions), and a lower penalty for inserting them. Insertions are slightly less likely than deletions. It has a higher penalty for A/G transition variants, and a similar penalty for C/T transition variants as other substitution penalties.

9 Map reads to the reference and convert to BAM format using *maf-convert* and *samtools*:

```
cat results/read_L50.txt | while read bc len;
  do echo ${bc} ${len};
  lastal -j 7 -P 10 -p plasmid.mat reference/${REFNAME}.fa \
      results/bestLong_100X_${bc}.fastq | last-split -n -m 0.99 | last-
postmask | \
    maf-convert sam | samtools view -h --reference reference/${REFNAME}.fa |
\
    samtools sort > results/${bc}_vs_${REFNAME}.bam;
  samtools index results/${bc}_vs_${REFNAME}.bam;
done
```

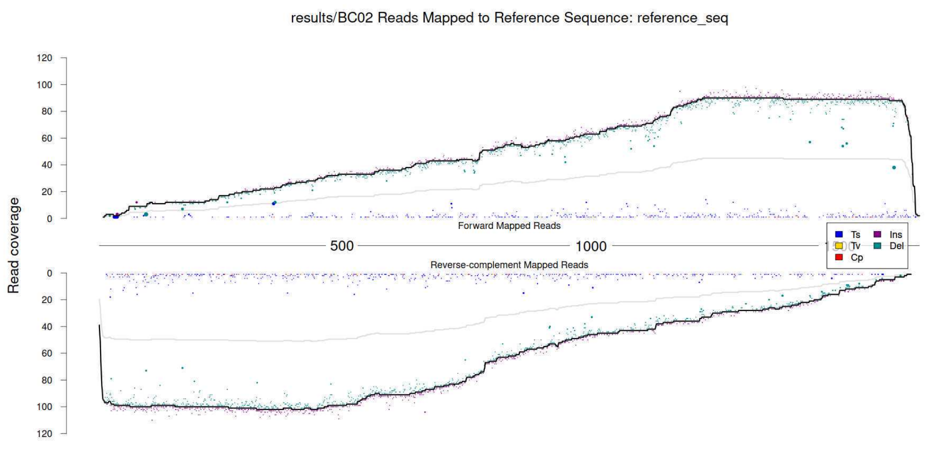
10 Split reads into forward and reverse-mapped sequences, and tally up base counts at each location:

```
cat results/read_L50.txt | while read bc len;
  do echo ${bc} ${len};
  samtools view -b -F 0x10 results/${bc}_vs_${REFNAME}.bam | \
    samtools mpileup --reference reference/${REFNAME}.fa -B -Q 0 - | \
    ~/scripts/readstomper.pl -c >
results/fwd_stomped_${bc}_vs_${REFNAME}.csv
  samtools view -b -f 0x10 results/${bc}_vs_${REFNAME}.bam | \
    samtools mpileup --reference reference/${REFNAME}.fa -B -Q 0 - | \
    ~/scripts/readstomper.pl -c >
results/rev_stomped_${bc}_vs_${REFNAME}.csv
done
```

11 Create a visualisation of the base-stomped output:

```
cat results/read_L50.txt | while read bc len;
  do echo ${bc} ${len};
  ~/scripts/stomp_plotter.r -f results/fwd_stomped_${bc}_vs_${REFNAME}.csv \
    -r results/rev_stomped_BC02_vs_${REFNAME}.csv -prefix results/${bc}
done
```

Example output (`*results/BC02_stompPlot_reference_seq.png*`):



Choice 2: De-novo Canu Assembly

12 Run a Canu assembly on each read set, using default options:

```
cat results/read_L50.txt | while read bc len;
  do canu -nanopore results/bestLong_100X_${bc}.fastq \
    -p ${bc} -d results/canu_${bc} genomeSize=${len};
done
```

13 Trim the assembled contigs based on Canu's trim recommendations:

```
cat results/read_L50.txt | while read bc len;
  do echo ${bc} ${len};
  samtools faidx results/canu_${bc}/${bc}.contigs.fasta \
    $(grep '^>' results/canu_${bc}/${bc}.contigs.fasta | \
      perl -pe 's/^>(.*?) .*trim=(.*)/$1:$2/') >
results/circTrimmed_${bc}.fasta;
done
```

14 After doing this, sometimes there are some samples that fail to assemble, especially if coverage is too low. For these samples, canu can be given a little bit more encouragement by fragmenting reads further. I have created a *normalise_seqlengths.pl* script that will fragment sequences so that they all have identical lengths, with an overlap length of at least a specified number of bases between the fragments. In this example, sequences are created that are 5000 bases in length (about 1/2 of the original plasmid length), with an overlap of 500 bases. The actual values for this will depend on the original plasmid length:

```
cp results/read_L50.txt results/read_L50.tricky.txt;
nano results/read_L50.tricky.txt # remove all except the difficult sequences
cat results/read_L50.tricky.txt | while read bc len;
   do echo ${bc};
   normalise_seqlengths.pl -fraglength 5000 \
      -overlap 500 <(zcat demultiplexed/reads_${bc}.fq.gz) >
results/fragmented_reads_${bc}.fastq;
done
```

Then re-run canu with the new reads, using the same settings as before:

```
cat results/read_L50.tricky.txt | while read bc len;
  rm -rf results/canu_${bc};
  do canu -nanopore results/fragmented_reads_${bc}.fastq \
    -p ${bc} -d results/canu_${bc} genomeSize=${len};
```

```
done
```

If everything worked, then the assembled contigs should now be available and trimmable:

```
cat results/read_L50.tricky.txt | while read bc len;
  do echo ${bc} ${len};
  samtools faidx results/canu_${bc}/${bc}.contigs.fasta \
    $(grep '^>' results/canu_${bc}/${bc}.contigs.fasta | \
      perl -pe 's/^>(.*?) .*trim=(.*)/$1:$2/') >
results/circTrimmed_${bc}.fasta;
done;
```