VERSION 9

MAR 13, 2024

DOI:
dx.doi.org/10.17504/protocols.io.36wgq4n5yvk5/v9

Protocol Citation: David A Eccles 2024. Plasmid Sequence Analysis from Long Reads. **protocols.io** https://dx.doi.org/10.17504/protocols.io.36wgq4n5yvk5/v9Version created by David A Eccles

Protocol status: Working
We use this protocol and it's working

Plasmid Sequence Analysis from Long Reads V.9

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)

ABSTRACT

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

For plasmid sample preparation for nanopore sequencing, see here. The most recent version of our rapid barcoding kit protocol at the time this protocol was written is attached below:

DAE_Rapid_Barcoding_Kit_Protocol_2… 218KB

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 and Dorado has some issues with chimeric reads, which can affect assembly.

Output(s):

- Consensus sequence per barcode as a fasta file

Created: Mar 13, 2024

Last Modified: Mar 13, 2024

PROTOCOL integer ID: 96611

Keywords: plasmid, nanopore, sequencing, rapid barcoding, data analysis, assembly

BEFORE START INSTRUCTIONS

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 $2}' demultiplexed/barcode_counts.txt);
  do ls demultiplexed/reads_${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 (bearing in mind that there may be issues with chimeric reads), 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

```
cd demultiplexed
mkdir results
```

Downloading Bioinformatics Scripts

3 This protocol uses scripts that I have developed over the course of my bioinformatics work to simplify fastq and fasta file processing. They can be downloaded from my gitlab repository:

```
git clone https://gitlab.com/gringer/bioinfscripts.git scripts
```

Read filitering

4 It is assumed that you have some prior knowledge of the plasmid length. If the length is unknown, it can be determined by looking at the L50 summary statistics produced from *fastx-length.pl* when run on a representative sample:

```
scripts/fastx-length.pl reads_BC05.fq.gz > /dev/null
```

Example output:

```
Total sequences: 8288
Total length: 29.360421 Mb
Longest sequence: 19.739 kb
Shortest sequence: 80 b
Mean Length: 3.542 kb
Median Length: 4.936 kb
N10: 410 sequences; L10: 5.051 kb
N50: 2747 sequences; L50: 5.007 kb    <==
N90: 5310 sequences; L90: 2.947 kb
A/T homopolymer lengths:
  Minimum: 0 bp
  Maximum: 1905 bp
  Mean: 13 bp
  Median: 6 bp
  Mode: 131 bp
  10th percentile: 0 bp
  90th percentile: 37 bp
Predicted / modeled quality across all reads:
  Minimum: q2
  Maximum: q20
  Mean: q12
  Median: q13
  Mode: q20
  10th percentile: q7
  90th percentile: q16
```

In this case, the plasmid length is approximately 5kb, so a system variable is created containing that information, as well as a value half that length:

```
plasmidLength=5000
plasmidHalfLength=2500
```

Filter out any reads that are less than half of the target read length, then from the remaining reads keep keep 100 of highest-quality reads. These reads are fragmented into 1.5kb pieces with 750bp overlap.

```
for bc in $(awk '{print $2}' barcode_counts.txt);
  do scripts/fastx-fetch.pl -lengthFilter ${plasmidHalfLength} -count 100
reads_${bc}.fq.gz | \
    ~/scripts/normalise_seqlengths.pl -fraglength 1500 -overlap 750 | \
    gzip > Filtered_reads_${bc}.fq.gz;
done
```

Choice 1: mapping to a reference sequence

5 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"
```

6 Create a LAST index for the reference sequence:

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

7 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 \
  Filtered_reads_BC*.fastq.gz
```

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.

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

```
for bc in $(awk '{print $2}' barcode_counts.txt);
  do echo ${bc};
  lastal -j 7 -P 10 -p plasmid.mat reference/${REFNAME}.fa \
      Filtered_reads_{bc}.fastq.gz | 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
```

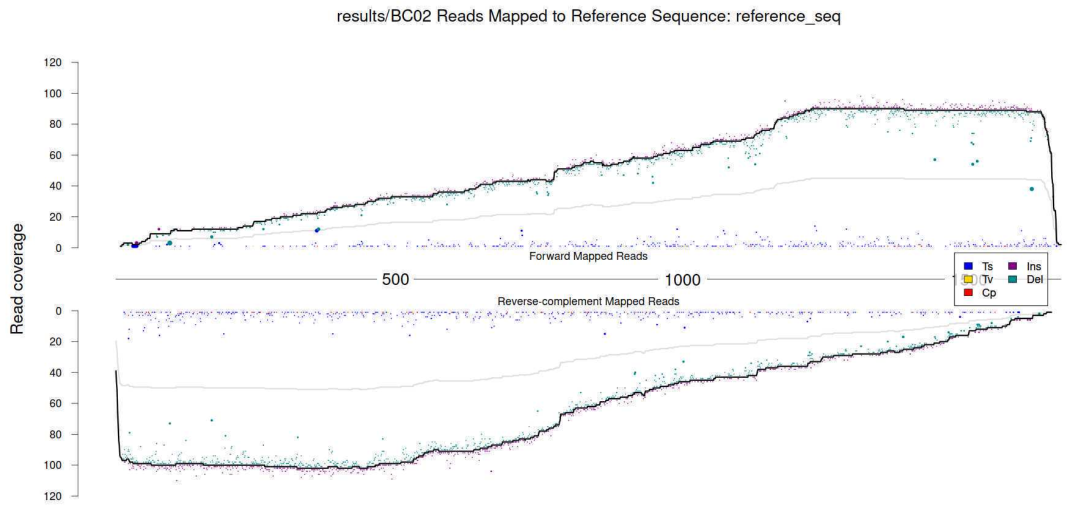
9 Split reads into forward and reverse-mapped sequences, and tally up base counts at each location:

```
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
```

10 Create a visualisation of the base-stomped output:

```
for bc in $(awk '{print $2}' barcode_counts.txt);
  do echo ${bc};
  ~/scripts/stomp_plotter.r -f results/fwd_stomped_${bc}_vs_${REFNAME}.csv \
    -r results/rev_stomped_${bc}_vs_${REFNAME}.csv -prefix results/${bc}
done
```

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



BC02 reads mapped to a reference sequence called 'reference_seq'

Choice 2: De-novo Canu Assembly

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

```
for bc in $(awk '{print $2}' barcode_counts.txt);
  do echo ${x};
  canu -nanopore Filtered_reads_${bc}.fq.gz -p ${bc} \
    -d results/canu_${bc} genomeSize=${plasmidLength};
done
```

12 Trim the assembled contigs based on Canu's trim recommendations for the longest assembled contig:

```
for bc in $(awk '{print $2}' barcode_counts.txt);
  do echo ${bc};
  samtools faidx results/canu_${bc}/${bc}.contigs.fasta
     $(grep '^>' results/canu_${bc}/${bc}.contigs.fasta | \
       perl -pe 's/^>(.*?) .*trim=([0-9]+)-([0-9]+)/$1.":".($2+1)."-".
($3)/e') | \
     scripts/fastx-fetch.pl  |  \
   scripts/fastx-sort.pl -l | scripts/fastx-fetch.pl -c 1 | \
   perl -pe "s/^>/>${bc}_/" > results/longest_circTrimmed_${bc}.fasta;
done
```

Final Cleanup - Orientation and Rotation

13 If plasmid sequences contain a long polyA region, the assembled sequences can be oriented assuming that the longest homopolymer is this polyA region:

```
for bc in $(awk '{print $2}' barcode_counts.txt);
  do echo ${bc};
  if scripts/fastx-hplength.pl results/longest_circTrimmed_${bc}.fasta |
tail -n 1 | grep "T";
    then scripts/fastx-rc.pl results/longest_circTrimmed_${bc}.fasta | \
      scripts/fastx-fetch.pl -oneline >
results/oriented_longest_circTrimmed_${bc}.fasta;
    else scripts/fastx-fetch.pl -oneline
results/longest_circTrimmed_${bc}.fasta \
      > results/oriented_longest_circTrimmed_${bc}.fasta;
  fi;
done
```

14 The plasmid can then be oriented based on a known alignment sequence.

First the alignment sequence is stored in a file 'align_seq.fa', and indexed using *lastdb*:

```
echo -e ">align_seq\nGAGACGGTCGACTGCGGCCGCTCGAGTCT" > align_seq.fa
lastdb -uRY4 -R01 align_seq.fa align_seq.fa
```

Then the alignment sequence is used to determine the rotation of the plasmid:

```
for bc in $(awk '{print $2}' barcode_counts.txt);
  do echo ${bc};
  scripts/fastx-rotate.pl \
    -o $(lastal align_seq.fa
results/oriented_longest_circTrimmed_${bc}.fasta | \
      grep BC | awk '{print $3}') \
    results/oriented_longest_circTrimmed_${bc}.fasta \
      > results/rotated_longest_circTrimmed_${bc}.fasta;
done
```