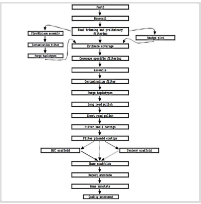

1

Jan 13, 2022

Long-read plant genome assembly and annotation: assembly V.1

Scott Ferguson[1], Ashley Jones[1], Justin Borevitz[1]

[1]Australian National University

1

protocol .

Scott Ferguson
Australian National University

With the introduction of long-read, third generation sequencing (e.g. Oxford Nanopore Technologies (ONT) and Pacific Biosciences (PacBio) and associated bioinformatics tools, we can now affordably assemble complex plant genomes with an accuracy and contiguity that was previously incredibly expensive to obtain. Long-read sequencing technology has dramatically improved our ability to generate de novo genome assemblies. In particular plant genomes, due to their large proportion of repetitive sequences, have highly benefited from long-read sequencing. Despite these improvements, challenges still remain in performing de novo assembly, namely in developing a reliable workflow and in tool choice. Here we present the *de novo* assembly proportion of our long-read plant genome assembly, scaffolding, and annotation workflow. We have developed and tested this workflow on many complex plant genomes. By using this workflow with a sufficient coverage of long-reads, a highly contiguous and accurate plant genome can be assembled.

Scott Ferguson, Ashley Jones, Justin Borevitz 2022. Long-read plant genome assembly and annotation: assembly. **protocols.io**
https://protocols.io/view/long-read-plant-genome-assembly-and-annotation-ass-b3r5qm86

protocol ,

Jan 13, 2022

Jan 13, 2022

56861

Data Input

1 De novo genome assembly works best if long-read sequencing has been performed. For best results, optimise a high-molecular weight DNA extraction, for example we use: High-molecular

weight DNA extraction, clean-up and size selection for long-read sequencing (https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0253830).

Perform long-read sequencing according to manufacturer's instructions and obtain sequence fast5 and (if provided) fastq files.

2 During the assembly of a genome we often want to count the number of sequences, and length of sequences within a fasta/q. The following script makes use of bioawk (bioawk will need to be in your path) to list all sequence names and sizes in a tab delimited file.
Save this script as gne-file.sh, make it executable (chmod u+x gne-file.sh), and place it in a location included in your path.

gne-file.sh

```
#!/bin/bash
echo $1
tmp=$(basename $1)
fileName=${tmp%.fast*}
bioawk -c fastx '{print $name, length($seq)}' $1 >
${fileName}.genome
```

Basecalling

3 While fastqs may be provided with your fast5 files, due to rapid improvements in the basecaller, it can be greatly beneficial to re-basecall. More recent basecallers, eg. guppy version 5, have a very low per-base error rate, resulting in more contagious assemblies.

Base call fast5 with guppy v5

```
## pick appropriate config file
#MinION/GridION + r9.4.1   = dna_r9.4.1_450bps_sup.cfg
#MinION/GridION + r10.3    = dna_r10.3_450bps_sup.cfg
#PromethION     + r9.4.1   = dna_r9.4.1_450bps_sup_prom.cfg
#PromethION     + r10.3    = dna_r10.3_450bps_sup.cfg
CFG=XXX.cfg

/path to guppy v5+/bin/guppy_basecaller \
   -c $CFG \
   --input_path /path to fast5 files/ \
   --save_path /path to save to/ \
   --records_per_fastq 0 \
   --device "cuda ID" \
   --recursive \
   --disable_pings \
   --disable_qscore_filtering
```

4 After basecalling, concatenate all fastq files into a single file and compress with gzip.

Concatenate all fastq files and gzip

```
find /path Guppy saved fastqs to/ -name ‘*.fastq’ | xargs cat | gzip > reads.fastq.gz
```

Read trimming and preliminary filtering

5 Your reads will contain sequencing adapters which require trimming. Examination of read quality has shown us that read ends, both 5` and 3`, are of low quality. Trimming additional sequence from read ends has the advantage of raising the average read quality, preserving as much of our read dataset as possible. Additionally, your fastq file may contain ONT DNA control strand reads which need to be identified and removed.

6 After read trimming, and in the same command queue, we remove all low quality reads and reads less than 1 Kbp in size.

Read trimming and preliminary filtering

```
zcat reads.fastq.gz |
   /path to Nano Pack/NanoLyse -r /path to DNA control
strand/DNA_CS.fasta | \
   /path to Nano Pack/NanoFilt --headcrop 200 --tailcrop 200 | \
   /path to Nano Pack/NanoFilt -q 7 -l 1000 | \
   gzip > preliminary-filter.fastq.gz
```

Genome size estimate

7 We next need an estimate of genome size so that read coverage can be calculated. If the genome size for your species or a closely related species is known, use this. If genome size is not known you will need to estimate your genome size using either the assembly method or Genome Scope.

Genome size estimate: Assembly method

8 Here we use a quick assembly method on a subset of our read data to assemble and get a genome size estimate. Randomly sample a subset (it is hard to guess how much of your dataset to use, we like to use 50% if sequencing was very successful or 75% if sequencing was less successful) of your read dataset and assemble with Flye (our preference) or Miniasm.

Randomly sample 50% of reads

```
seqtk sample -s10 preliminary-filter.fastq.gz $(echo $(zgrep '^>'
preliminary-filter.fastq.gz |wc -l)/2 | bc) | gzip > subset.fastq.gz
```

Randomly sample 75% of reads

```
seqtk sample -s10 preliminary-filter.fastq.gz $(echo $(zgrep '^>'
preliminary-filter.fastq.gz |wc -l)*0.75 | bc) | gzip > subset.fastq.gz
```

Assemble subset of reads with Flye

```
threads=XX

path to flye/flye --nano-raw subset.fastq.gz --out-dir /path to save assembly to/ --threads $threads
```

9 You wont be able to get a good estimate of your genome size until you remove any contaminate contigs and purge haplotigs from your assembly.

To filter contamination see step 16.
To purge haplotigs see step 17.

Genome size estimate: Genome Scope (K-mer)

10 If you have access to high accuracy reads (eg. illumina, or HiFi) you can use the k-mer based genome size estimator Genome Scope 2.0. Polishing reads may be used here but may fail if read coverage is insufficient. Additionally, this method will give an estimated ploidy.

Genome Scope 2.0

```
threads=XXX

jellyfish count -C -m 21 -s 1000000000 -t $threads accurate-reads.fastq.gz -o reads.jf
jellyfish histo -t $threads reads.jf > reads.histo
genomescope.R -i reads.histo -o / path to output/ -k 21
```

Coverage specific filtering

11 Now that we have an estimated genome size we can calculate our estimated coverage and perform additional read filtering if our read dataset has coverage in excess of 60x. More reads, higher coverage, can be used if you have a lot of compute time. More coverage will assemble a better genome, but the effects of increased coverage plateau around ~80x. If using a lot of coverage (>60x) assembly settings can be tweaked to keep the best reads after read correction.

$$coverage = \frac{total\ number\ of\ bases\ in\ read\ set}{genome\ size}$$

12 Additional read filtering occurs by two metrics, average read quality scores and read length. It can be hard to find the perfect intersection of these two variables, we tend to filter more on length than on quality so as to keep very long reads within our dataset. Change the length and quality values to suit your dataset.

Removing more reads here will result in faster read correction and trimming in Canu. But it may be preferable to input a larger dataset into Canu and use Canu's "corOutCoverage" parameter to reduce data after read correction. When coverage is greatly in excess we perform additional filtering to ~60-80x and use Canu's corOutCoverage parameter to further reduce coverage to ~30-50x, depending on minimum read length.

Length only filtering

```
gne-file.sh preliminary-filter.fastq.gz
for len in 2000 5000 10000 20000
do
   awk -v L=$len '{if($2 >= L){print $1}}' preliminary-filter.genome > use.lst
   seqtk subseq preliminary-filter.fastq.gz use.lst | gzip > ${len}_final.fastq.gz
   gne-file.sh ${len}_final.fastq.gz
done
```

Quality only filter

```
for qual in 8 9 10
do
   zcat all.fasta.gz | /path to Nano Pack/NanoFilt -q $qual | gzip > q${qual}.fastq.gz
   gne-file.sh q${qual}.fastq.gz
done
```

Filter on quality and length

```
for qual in 8 9 10
do
   zcat all.fasta.gz | /home/801/sf3809/.local/bin/NanoFilt -q $qual | gzip > q${qual}.fastq.gz
   gne-file.sh q${qual}.fastq.gz
   for len in 2000 5000 10000 20000
   do
      awk -v L=$len '{if($2 >= L){print $1}}' q${qual}.genome > use.lst
      seqtk subseq q${qual}.fastq.gz use.lst | gzip > q${qual}-${len}_final.fastq.gz
      [ -e q${qual}-${len}_final.fastq.gz ] && rm q${qual}-${len}_final.fastq.gz || gne-file.sh q${qual}-${len}_final.fastq.gz
   done
done
```

13 Having filtered your read dataset we now need to find which filtered fastq to use. Using the genome file created for each fastq, we first calculate the total amount of sequence within each dataset and finally calculate the estimated coverage. You will need to determine and pick the dataset with the most suitable coverage.

Calculate coverage for all filtered datasets

```
genomeSize=XXX
for gne in *.genome
do
   filt=$(basename $gne .genome)
   echo $filt $(echo $(cut -f2 $gne | paste -s -d+ | bc)/${genomeSize} | bc)
done
```

Successful assembly will result in the creation of ${label}.contigs.fasta

Assemble genome

14 Many of the parameters and methods of getting Canu to work are workspace and dataset dependent. We will detail the parameters that we have found to impact assembly the most, or

have been necessary to get Canu to assemble our genomes in a timely manner.

Canu's read the docs website contains a lot of good information on assembly.

corOutCoverage - how much coverage to keep after read correction. Canu will estimate corrected read size before correcting and only correct the longest X coverage of reads. A larger value can give a better assembly, but will also consume many more computer resources.
Another consideration when setting this parameter is read length. If your read set is very long (min size >> 20kb) assembly can be very slow, as such a reduced amount of corrected reads may be necessary.
This should be tuned to suit your compute resources. The maximum recommended is 80, with 30-60 producing very good results.

correctedErrorRate - The allowable percent error rate between corrected reads (eg. 0.12 = 12%). Meta setting for many other assembly stages. The default is 0.144 for ONT reads, if you have excessive input coverage reduce this value. If starting with low coverage leave this at defaults. If using high accuracy ONT reads (basecalled with guppy 5+) Canu docs recommend to set at 0.105.

corMaxEvidenceErate - Only correct reads that have this fraction or lower fraction of errors, determined by overlapping reads. We have found this setting to be necessary for many plant (highly repetitive) genome assemblies which fail to assemble as jobs exceed our maximum allowed run time. We leave this set at 0.15 for all assemblies.

batOptions - The settings used here are described within the canu documents as "avoid collapsing the genome". We have found that assemblies made with these settings have a broader distribution of contigs lengths, ie. more short and more long contigs, and as the majority of small contigs are removed during haplotype purging, these parameters give a more contiguous final genome. We leave this as "batOptions=-dg 3 -db 3 -dr 1 -ca 500 -cp 50" for all assemblies. The small purged contigs are regions of the parental haplotype in the alternative parent.

fast - Use MHAP (a fast but less accurate aligner) for read correction, trimming, and assembly (default: correction = MHAP, trimming and assembly = ovl). With some plant genomes read overlapping would fail to complete in a timely manner, as such we use fast (MHAP) for all overlapping to avoid wasting large amounts of our compute allocation.

Assemble genome with Canu

```
label=example
saveLocation="/location to save to/"
reads="/location of reads/example.fastq.gz"
stageDirectory="/path to fast disk space/"
genomeSize=500m
correctCoverage=60
correctedErrorRate=0.105
corMaxEvidenceErate=0.15

/path to canu/canu \
        -p $label \
        -d ${saveLocation}/canu_${label} \
        -nanopore $reads \
        genomeSize=${genomeSize} \
        stageDirectory=${stageDirectory} \
        corOutCoverage=${correctCoverage} \
        "batOptions=-dg 3 -db 3 -dr 1 -ca 500 -cp 50" \
        correctedErrorRate=${correctedErrorRate} \
        corMaxEvidenceErate=${corMaxEvidenceErate} \
        -fast
```

15 Successful assembly will result in the creation of ${label}.contigs.fasta

Contamination filter

16 DNA extraction likely resulted in the collection and sequencing of DNA from contaminate sources (eg. fungus living on leaf surfaces). If these sequences were assembled they will be present within your assembly. Contamination sequences are taxonomically identified with blast and blob tools, and removed using seqtk and bash commands.

Setup

```
genome=/path to assembly/${label}.contigs.fasta or unitigs.fasta
reads="/location of reads/example.fastq.gz"
cpus="XX"
memory="XXG"
ncbiDB="/path ot ncbi db/nt"
```

Align reads to genome

```
minimap2 -t $cpus -ax map-ont $genome $reads | \
   samtools sort -m $memory -@ $cpus > sorted_mapped_reads.bam
samtools index -@ $cpus sorted_mapped_reads.bam
```

Blast contigs/unitigs

```
blastn -query $genome -db $ncbiDB -outfmt "6 qseqid staxids bitscore std" -max_target_seqs 10 -max_hsps 1 -evalue 1e-25 -out ${label}.out
```

Blobtools - id sequence taxonomy

```
blobtools create -i $genome -b sorted_mapped_reads.bam -t ${label}.out -o contamination
blobtools plot -i contamination.blobDB.json -o blobP
blobtools view -i contamination.blobDB.json -o blobV
```

Create contamination filtered fasta

```
grep -v '^#' blobV.contamination.blobDB.table.txt | grep -vE
'Bacteroidetes|Ascomycota|Proteobacteria|Arthropoda' | awk '{print
\$1}' > keep.lst
grep -v '^#' blobV.contamination.blobDB.table.txt | grep -E
'Bacteroidetes|Ascomycota|Proteobacteria|Arthropoda'  | awk '{print
\$1}' > contaminant.lst
seqtk subseq $genome keep.lst > keep.fasta
seqtk subseq $genome contaminant.lst > contamination.fasta
```

Purge haplotigs

17 When regions of the parental haplotypes highly differ the assembler will produce two sequences that represent the same genetic region. As we are building a pseudo-haploid genome we need to identify the sequences that make up these doubly represented regions and remove one. Haplotig removal is performed to improve genome contiguity, as such the longest sequence is kept within the assembly. As this process aligns assembly reads to our genome we can also identify plasmid sequences within our assembly as they will have a very high or very low read coverage.

Note: read alignment was already performed during contamination removal, as such we skip here and use the existing bam file.

Setup

```
bam="/path to contamination filtering/sorted_mapped_reads.bam"
genome="/path to contamination filtering/keep.fasta"
cpus=XX
```

Build read histogram

```
purge_haplotigs hist -b $bam -g $genome -t $cpus
```

View read histogram (sorted_mapped_reads.bam.histogram.png) and pick cutoffs
for more information see https://bitbucket.org/mroachawri/purge_haplotigs/src/master/

Purge genome

```
lower=XX
medium=XX
high=XX

purge_haplotigs contigcov -i sorted_mapped_reads.bam.gencov -l $lower -m $medium -h $high
purge_haplotigs purge -g $genome -c coverage_stats.csv -b $bam -d -t $cpus
```

Long read polish

18 Despite Canu's read correction stage, polishing of your assembly with assembly read fastq has been shown to improve base accuracy. Improvement was assessed with BUSCO scores. Long read polishing is performed with minimap2 and Racon. We typically perform two rounds of polishing in this step.

Racon polish

```
genome="/path to contamination filtering/curated.fasta"
reads="/location of reads/example.fastq.gz"
cpus="XX"
rounds=2

for ((i=1; i<=${rounds};i++))
do
   # map long reads with minimap2
   minimap2 -t $cpus -ax map-ont ${genome} ${reads} > mapped_reads.sam
   # run racon
   racon -t $cpus -u ${reads} mapped_reads.sam ${genome} > racon-${i}
   genome=racon-${i}.fasta
done
```

Short read polishing - NextPolish

19 The first of our short read polishers is Next Polish, it works with bwa mem. For this polishing step we use 20-25x coverage of illumina data. There are a number of different ways of running Next polish, which one you use will depend on your compute environment. We have found success within our compute environment by running the python3 script nextpolish1.py.
We typically perform two rounds of polishing in this step.

Nextpolish polish

```
reads="/location of reads/example.fastq.gz"
genome="/path to long read polish/racon-2.fasta"
rounds=2
cpus=XX
memory=XXg

for ((i=1; i<=${rounds};i++))
do
#step 1
    #index the genome and do alignment
    bwa index $genome
    bwa mem -t $cpus -p $genome $reads \
        | samtools view -@ $cpus -F 0x4 -b - \
        | samtools fixmate -m -@ $cpus  - - \
        | samtools sort -m $memory -@ $cpus - \
        | samtools markdup -@ $cpus -r - sorted.bam
    #index bam and genome
    samtools index -@ $cpus sorted.bam
    samtools faidx $genome
    #polish genome
    python nextpolish1.py -g $genome -t 1 -p $cpus -s sorted.bam >tmp~${i}.fasta
#step 2
    #index the genome and do alignment
    bwa index tmp~${i}.fasta
    bwa mem -t $cpus -p tmp~${i}.fasta ${readFile} \
        | samtools view -@ $cpus -F 0x4 -b - \
        | samtools fixmate -m -@ $cpus  - - \
        | samtools sort -m $memory -@ $cpus - \
        | samtools markdup -@ $cpus -r - sorted.bam
    #index bam and genome
    samtools index -@ $cpus sorted.bam
    samtools faidx tmp~${i}.fasta
    #polish genome
    python nextpolish1.py -g tmp~${i}.fasta -t 2 -p $cpus -s sorted.bam > nextPolish~${i}.fasta

     genome=nextPolish~${i}.fasta
done
```

Short read polishing - Hapo-G

20 In addition to Nextpolish we run a second short read polisher, Hapo-G. Results obtained within the Hapo-G paper indicate that polishing first with Next Polish and then Hapo-G obtained the most accurate genome.
We typically perform three rounds of polishing in this step.

Haop-G polish

```
reads="/location of reads/example.fastq.gz"
genome="/path to Next Polish/racon-2.fasta"
rounds=3
cpus=XX
memory=XXg

zcat $genome | sed -e 's/_.*//g' > hapog-0.fasta
genome=hapog-0.fasta

for ((i=1; i<=${rounds};i++))
do
   #index the genome and do alignment
   bwa index $genome
   bwa mem -t $cpus -p $genome $reads \
      | samtools sort -m $memory -@ $cpus > sorted.bam
   #index bam and genome
   samtools index -@ $cpus sorted.bam

   #polish
   python3 hapog.py --genome $genome -b sorted.bam -o ${i} -t $cpus
-u

   genome=${i}/hapog_results/hapog.fasta
done
```

Filter small contigs

21 Small contigs contain little useful genetic information and are filtered from our assemblies. We typically remove any sequence less than 1 Kbp in length.

Filter small contigs

```
bioawk -c fastx 'length($seq) > 1000{print ">" $name "\n" $seq}'
hapog.fasta > sizeFiltered.fasta
```

Filter plasmid contigs

22 Despite filtering for plasmid sequences during haplotig purging some plasmid sequences may still remain within the assembly. Here we use MUMmer to find alignments between our genome and known plasmid sequences. The sequences we screen our genome with are obtained from a public repository such as the NCBI. If no plasmid sequences exist for your species, find the closest available relative and download chloroplast and mitochondria sequences.

MUMmer align and dotplot plasmid seqeunces

```
genome=”/path to genome/sizeFiltered.fasta”
for plasmid in /path_to_plasmid/XX.fasta /path_to_plasmid/YY.fasta
do
   tmp=$(basename $plasmid)
   name=${tmp%.fasta*}

   mkdir $name
   cd $name

   nucmer -t 1 $genome $plasmid
   delta-filter -1 -m -i 90 -l 100 out.delta > out.1delta

   for tig in `grep '^>' out.1delta | cut -f1 -d' ' | sort | uniq | sed -e 's/>//g'`
   do
      echo $tig > tmp.lst
      seqtk subseq $genome tmp.lst > tmp.fasta
      nucmer -t 1 tmp.fasta $plasmid -p $tig
      delta-filter -1 ${tig}.delta > out.1delta
      mummerplot --fat --png --large out.1delta -p ${tig}
   done
   rm tmp.lst tmp.fasta
   cd ../..
done
```

23 On completion of MUMmer alignments you will find a number of png files within subdirectories named after your plasmid sequences. These images contain a dot plot showing the alignment between all sequences within your genome that are highly aligned to the plasmid. Each image needs to be reviewed and a decision made on whether the sequence originates within the plasmid or not. If you decide that the sequence is plasmid, record its name in remove.lst.

24 Once you have decided which sequences you wish to remove the following script will perform the sequence removal.

Filter plasmid sequences

```
genome=”/path to genome/sizeFiltered.fasta”

bioawk -c fastx '{print $name}' $genome > all.lst
grep -vwf remove.lst < all.lst > keep.lst
seqtk subseq $genome keep.lst > plasmid-filtered.fasta
```

Genome complete

25 Your genome is now complete and ready for use. If possible it is highly recommended to scaffold your genome using Hi-C or if Hi-C is not available with synteny information obtained from a closely related species. If you wish to scaffold please see the section on scaffolding. Also, see the section on assembly quality to help with assessing your assembly success.