



1

Jan 13, 2022

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

Scott Ferguson¹, Ashley Jones¹, Justin Borevitz¹

¹Australian 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 assembly scaffolding 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: scaffolding. **protocols.io**
<https://protocols.io/view/long-read-plant-genome-assembly-and-annotation-sca-bw45pgy6>



protocol ,

Aug 04, 2021

Jan 13, 2022

52093

Genome scaffolding

- 1 Currently your genome will not exist as full chromosomes, rather as fragmented sections of chromosomes. To increase the utility of your genome scaffolding is performed. Scaffolding

attempts to find how your sequences should be joined to form full chromosomes and joins them. There are a number of ways of scaffolding genome assemblies, here we focus on a synteny based approach and Hi-C.

Hi-C is the recommended method, but obtaining good quality Hi-C data can be difficult and expensive.

Hi-C: Quality control

- 2 The first step in scaffolding with Hi-C is to determine if your Hi-C provides sufficient information for scaffolding. The two tools we like to use to determine library quality are `hic_qc` from Phase Genomics and Juicer.

Phase genomics QC report (pdf)

```
label="XXX"
```

```
R1="/path to reads/XX_R1.fastq.gz"
```

```
R2="/path to reads/XX_R2.fastq.gz"
```

```
genome="/path to genome/plasmid-filtered.fasta"
```

```
cpus=XX
```

```
# first align Hi-C reads to your genome
```

```
mkdir ${label}~phase
```

```
cd ${label}~phase
```

```
bwa index $genome
```

```
bwa mem -t $cpus -5SP $genome $R1 $R2 > aligned.sam
```

```
cat aligned.sam | samblaster > tmp.sam
```

```
samtools view -@ $cpus -S -h -b -F 2316 tmp.sam > blaster.bam
```

```
rm *.sam
```

```
# Now generate QC report
```

```
python hic_qc.py -b blaster.bam -o blaster
```

Phase QC: generates a pdf report. See <https://phasegenomics.github.io/2019/09/19/hic-alignment-and-qc.html> to aid in reading of report

Juicer QC

```
label="XXX"
R1="/path to reads/XX_R1.fastq.gz"
R2="/path to reads/XX_R2.fastq.gz"
genome="/path to genome/plasmid-filtered.fasta"
cpus=XX
enzyme="DpnII"
juicerPath=""

mkdir ${label}~juice
mkdir ${label}~juice/fastq
cd ${label}~juice

ln $R1 fastq/
ln $R2 fastq/
ln $genome .

python3 ${juicerPath}/misc/generate_site_positions.py $enzyme
$label $(basename fnaFile)
bwa index $fnaFile
gne-file.sh $genome

${juicerPath}/scripts/juicer.sh -g $label -z $genome -p $(basename
$genome .fasta).genome -y ${label}_${enzyme}.txt -D ${juicerPath}
-t ${PBS_NCPUS}
${juicerPath}/CPU/common/cleanup.sh
```

Interpretation of Hi-C quality results can be difficult. The main things we are looking for are:

- A large number of sequenced read pairs
- A large number of inter-chromosomal read pairs
- A large number of intra-chromosomal read pairs
- Low percent of PCR duplicates
- Low percent of unmapped reads
- Phase Genomics library statistic (Phase Genomics website contains details about these results)

Hi-C: Scaffolding with 3D-DNA

- 3 Your Hi-C library contains adequate read pairs to anchor sequences and build pseudo-chromosomes; you are now ready to scaffold. 3D-DNA runs in two parts: initial scaffolding is done and results can be viewed and modified/fixed by the user, before a final run of 3D-DNA produces a final scaffolded genome. 3D-DNA requires output from Juicer.

3D-DNA has a lot parameters that can be user configured, and improve scaffolding results. The best place to find information and help on how to set these can be found here: <https://groups.google.com/g/3d-genomics> and the Genome Assembly Cookbook linked here: <https://github.com/aidenlab/3d-dna>

3D-DNA: initial scaffolding

```
nodups="/path to juicer output/aligned/merged_nodups.txt"  
genome="/path to genome/plasmid-filtered.fasta"  
minSeqSize=1000  
DNAPath=""  
  
bash ${DNAPath}/run-asm-pipeline.sh -i $minSeqSize $assembly  
$nodups
```

4 3D-DNA will generate two files of interest:

- XX.rawchrom.assembly
- XX.rawchrom.hic.

These two files can be viewed in Juicebox. Juicebox will show you how your sequences have been ordered and rotated to make your pseudo-chromosomes and show the evidence use for the joining in the form of a heat map (see <https://github.com/aidenlab/Juicebox/wiki> for help). After editing (or not) your Hi-C results save your new hic and assembly files and proceed to finalising your genome scaffolding with 3D-DNA.

3D-DNA: Finalise scaffolding

```
run-asm-pipeline-post-review.sh -r juicebox_edited.assembly /path to  
juicer output/aligned/merged_nodups.txt
```

5 After scaffolding with Hi-C you can also scaffold with synteny using a close relative, as per steps 7-8. Whether you wish to do this or not will depend on the quality of the syntenic reference, phylogenetic distance, and how successful your Hi-C scaffolding was. This process will not break up your scaffolded sequences, only order and orient them according to how they align to the scaffolding reference.

A additional benefit of this method is that it will name your scaffolds according to the naming scheme established from the scaffolding reference, i.e. if the scaffolding reference has

chromosomes 1 to 4, your syntenic scaffolded reference will also contain sequences called chromosome 1 to 4.

Scaffold: Synteny

- 6 If Hi-C, genetic maps or other data is not available to anchor sequences together and create pseudo-chromosomes synteny information from a closely related species can be used. The tool we prefer to use here is RagTag (formally RaGOO) which uses minimap2 to align your contigs against the scaffolding genome, and anchor and orient the contigs into pseudo-chromosomes.

The first step in scaffolding is to remove all unplaced sequences that exist within the scaffolding reference, i.e. remove sequences that are not part of a chromosome.

Get list of sequence names in syntenic reference

```
scaffoldGenome="/path to scaffolding genome/sppXXX.fasta"  
genome="/path to genome/plasmid-filtered.fasta"
```

```
bioawk -c fastx '{print $name}' $scaffoldGenome > scaffold.lst
```

- 7 Open scaffold.lst in a text editor (eg. nano) and remove all sequence names that are not chromosomes, ie. reduce scaffolds.lst to only chromosomes. Once you have a list of chromosome sequence names scaffolding is performed.

RagTag will append "_ragtag" to the end of scaffold names, we use sed to remove this.

Scaffold genome

```
cpus=XXX
```

```
seqtk subseq $scaffoldGenome scaffold.lst > reference.fasta  
python3 ragoo.py -m /path to minimap2/minimap2 -C -t $cpus $contigs  
reference.fasta
```

```
sed -e 's/_ragtag//g' ragoo_output/ragoo.fasta > scaffold.fasta  
rm scaffold.lst reference.fasta
```