1. Merged individual fastq files from minion basecaller output into a single file.
2. QC (FastQC)
3. Trim adapters with Porechop (v). Discard reads containing middle adapters.

porechop -i reads.fastq -o porechopped.fastq \

--threads 42 --discard\_middle

1. Quality and length filtering with NanoFilt (v). Headcrop 50bp, filter with q10 quality threshold

NanoFilt -q 10 -l 500 --headcrop 50 \

porechopped.fastq > porechopped\_nanofilt\_q10.fastq

1. Postprocessing QC (FastQC).
2. Collapsed haplotype / haploid / mosaic / consensus assembly methods:
   1. Canu:
      1. Canu-correct (with –corOutCoverage higher than our read depth to correct all reads)

canu -correct -p canu\_corrected -genomeSize=12m \

-corOutCoverage=1000 -d canu\_corrected -nanopore \

preprocessed\_reads.fastq

* + 1. Canu-trim

canu -trim -p canu\_corrected\_trimmed \

-d canu\_trimmed genomeSize=12m \

-nanopore-corrected canu\_corrected.fasta

* + 1. Canu-assemble

canu -p canu\_sbay -d canu\_asm \

genomeSize=12m correctedErrorRate=0.039 \

-trimmed -corrected -nanopore canu\_corrected\_trimmed.fasta

* 1. Flye:

flye --nano-raw preprocessed\_reads.fastq --threads 42 \

--out-dir flye\_asm --iterations 2

* 1. Minasm:

minimap2 -x ava-ont -t8 | gzip -1 > reads.paf.gz

miniasm -f preprocessed\_read.fastq minimap.paf.gz > miniasm.gfa

awk ’/^S/{print “>”$2”\n”$3}’ miniasm.gfa > miniasm.fasta

* 1. Necat

Default parameters after setting up NECAT configuration files

* 1. Nextdenovo:

Prepare input.fofn file (with path to single concatenated .fastq file)

Create run.cfg with read type, paths, etc.

nohup nextDenovo run.cfg &

* 1. Ra

ra -t 48 -x ont preprocessed\_reads.fastq > ra.fasta

* 1. Raven

raven -t 16 preprocessed\_reads.fastq > raven.fasta

* 1. Wtdbg2

wtdbg2 -x ont -g 12m -i preprocessed\_reads.fastq \

-t 32 -fo wtdbg2\_asm

wtpoa-cns -t 32 -i wtdbg2\_asm ctg.lay.gz -fo wtdbg2\_asm.raw.fa

minimap2 -t 32 -ax map-ont -r2k \

wtdbg2\_asm.raw.fa preprocessed\_reads.fastq | samtools sort \

-@4 > wtdbg2\_asm.bam

samtools view -F0x900 wtdbg2\_asm.bam | ./wtpoa-cns -t 16 \

-d wtdbg2\_asm.raw.fa -i - -fo wtdbg2\_asm.cns.fa

* 1. Canu- SMARTdenovo (assemble Canu corrected and trimmed reads with SMARTdenovo)

Used SMARTdenovo’s internal scripts with Canu corrected and trimmed output

1. Haplotype-aware assembly methods (both haplotype-aware assembly and trio binning)
   1. Canu-trio (Using Illumina reads from S. eubayanus and S. uvarum

canu -p sbay\_canu\_trio -d sbay\_canu\_trio genomeSize=12m \

-haplotypeE eubayanus.fastq -haplotypeU uvarum.fastq \

-corOutCoverage=500 -nanopore corrected\_and\_trimmed\_reads.fastq

* 1. Flye (with –keep haplotypes)

flye --nano-raw preprocessed\_reads.fastq --threads 30 \

--out-dir flye\_keep\_haplotypes --iterations 2 --keep-haplotypes

* 1. Shasta

Attempted many assemblies with various parameters, but we were unable to generate assemblies that were not heavily fragmented assemblies (hundreds of contigs) or with expected genome sizes. For this reason, we did not consider Shasta assemblies in downstream analysis.

1. Polish assemblies

As none of the haplotype-aware assemblies produced a high-quality draft genome (in terms of N50, contig number, or expected size), they were left our of further analysis. The assemblies generated in 6) above were polished by 4 rounds of racon and one round of Medaka prior to gene prediction and genome completeness assessment (BUSCO), as research suggests that both are highly improved post-polishing.

* 1. Racon (four rounds for each assembly)

BASENAME=canu\_asm

ln -is $BASENAME.fasta $BASENAME.racon0.fasta

for n in 1 2 3 4;

do

minimap2 $BASENAME.racon`expr $n - 1`.fasta preprocessed\_reads.fastq > minimap.paf;

racon preprocessed\_reads.fastq minimap.paf $BASENAME.racon`expr $n - 1`.fasta -e 0.15 -t 30 -m 8 -x -6 -g -8 -w 500 > $BASENAME.racon$n.fasta;

done

* 1. Medaka (once for each racon-polished assembly)

medaka\_consensus -i preprocessed\_reads.fastq -d {racon\_polished\_assembly}.fasta -o {assembly}.medaka.fasta -t 48

1. Quast / BUSCO / GeneMark-ES / Augustus (for each of the assemblies in 6) above).

quast.py --fungus -f -o ./ -t 16 -b ../{polished\_assembly}.fasta

1. Phase haplotype assemblies
   1. Pepper-Margin-Deepvariant

Internal scripts (set up path structure, otherwise used default parameters)

* 1. WhatsHap
     1. Generate sorted bam (minimap2)

minimap2 -a -x map-ont {polished\_assembly}.fasta \

preprocessed\_reads.fastq > {assembly}.aln.sam

samtools view -bS {assembly}.aln.sam > {assembly}.aln.bam

samtools sort -o {assembly}.aln.sorted.bam {assembly}.aln.bam

samtools index {assembly}.aln.sorted.bam

* + 1. Variant calling (Freebayes)

freebayes -f {polished\_assembly}.fa \

{assembly}.aln.sorted.bam > {assembly}.vcf

* + 1. Phasing with WhatsHap

whatshap phase -o {assembly}.phased.vcf \

--reference={assembly}.fasta {input}.vcf {input}.bam

# Generate haplotagged bam to assess quality of phasing

# And for visualization with IGV

whatshap stats --gtf=phased.gtf phased.vcf

bgzip phased.vcf

tabix phased.vcf.gz

whatshap haplotag -o haplotagged.bam --reference nextdenovo.fasta phased.vcf.gz mapped.sorted.bam --ignore-read-groups

1. Generate haplotype-level fasta
   1. Bfctools consensus

tabix {assembly}\_pmd.vcf.gz

bcftools consensus -H 1 -f {polished assembly}.fasta \

{phased assembly\_vcf}.gz > {assembly}\_hap1.fasta

bcftools consensus -H 2 -f { polished assembly }.fasta \

{phased assembly\_vcf}.gz > {assembly}\_hap2.fasta

1. The sppIDer program requires two main steps in order to determine hybridization. The first step involves the combineRefGenomes.py script, which takes all parent reference genome assemblies and combines them into one file to be used later.  All parent genome assemblies came from NCBI. combineRefGenomes.py requires the genome .fasta files to have gene names in a specific format: “name of species - name of gene,” which is not standard format for genome assemblies coming from NCBI. To correct this, the following python script was used.

from Bio import SeqIO

original\_file = "input.fasta"

corrected\_file = "output.fasta"

species\_name = "species\_name"

with open(corrected\_file, 'w') as corrected:

   for seq\_record in SeqIO.parse(original\_file, "fasta"):

       print(seq\_record.id)

       print(seq\_record.description)

       seq\_record.id = species\_name + "-" + seq\_record.id

       seq\_record.description = seq\_record.id

       SeqIO.write(seq\_record, corrected, "fasta")

Once the formatting of each parent genome .fasta file was corrected, combineRefGenomes.py was ran to combine all the parent genomes. This required a KEY.txt file listing the species name and corresponding .fasta file name, tab separated. sppIDer was ran from a Docker image with the following docker command:

 docker run \

--rm -it \

--mount type=bind,src=$(pwd),target=/tmp/sppIDer/working \

--user "$UID:$(id -g $USERNAME)" \

glbrc/sppider \

  combineRefGenomes.py

  --out REF.fasta \

  --key KEY.txt

combineRefGenomes.py outputs a REF.fasta file, needed as input for the next step, as well as six other versions of the REF.fasta file for downstream use. The next step for sppIDer is the actual sppIDer.py script, which runs a bwa to map reads from our genome assembly to the reference parent genomes, and then runs outputs through R to plot percentages and MQ scores. sppIDer.py requires the REF.fasta file generated in the previous step, as well as the .fastq file(s) from our assembly. sppIDer.py was run using the following docker command:

docker run \

--rm -it \

--mount type=bind,src=$(pwd),target=/tmp/sppIDer/working \

--user "$UID:$(id -g $USERNAME)" \

glbrc/sppider \

  sppIDer.py \

  --out OUT \

  --ref REF.fasta \

  --r1 our\_genome\_assembly.fastq \