**Introduction**

This is a pipeline for high-quality *de nove* genome assembly and accurate gene annotation of vertebrate.

**Installation**

git clone <https://github.com/zhengchangsulab/A-genome-assebmly-and-annotation-pipeline.git>

export PATH=$PATH:/PATH/TO/A-genome-assembly-and-annotation-pipeline/bin/

export PATH=$PATH:/PATH/TO/ A-genome-assembly-and-annotation-pipeline/scripts/

**Running the pipeline**

**Genome assembly**

Dependencies

* Wtdbg2
* Samtools
* Minimap2
* BWA
* SALSA
* Bedtools
* Python2
* PBJelly
* Racon
* Nextpolish
* Python3

Preparation

* Illumina paired-end sequencing reads
* PacBio/Nanopore long reads
* Hi-C paired-end reads

Step 1

Generate contigs using long reads and polish the contigs using Illumina short reads using Wtdbg2. Given the nanopore long reads with 109X and 1Gbp estimated genome size, we used the following command lines:

NANOPORE=nanopore long reads.fastq

SHORTREAD1=Illumina paired-end-1.fastq

SHORTREAD2=Illumina paired-end-2.fastq

PREFIX=F025

threads=32

wtdbg2 -x ont -g 1g -X 109 -e 10 -i Nanopore long reads.fastq -t $threads -fo $PREFIX

wtpoa-cns -t $threads -i $PREFIX\.ctg.lay.gz -fo $PREFIX\.ctg.fa

minimap2 -t $threads -ax map-ont $PREFIX\.ctg.fa $NANOPORE | samtools view -Sb | samtools sort -@ $threads-1 - > $PREFIX\.ctg.crt.bam

samtools view -F 0x900 $PREFIX\.ctg.crt.bam | wtpoa-cns -t $threads -d $PREFIX\.ctg.fa -i - -fo $PREFIX\.ctg.2nd.fa

bwa index $PREFIX\.ctg.2nd.fa

bwa mem -t $threads $PREFIX\.ctg.2nd.fa $SHORTREAD1 $SHORTREAD2 | samtools sort -@ $threads-1 -O SAM | wtpoa-cns -t $threads -x sam-sr -d $PREFIX\.ctg.2nd.fa -i - -fo $PREFIX\.ctg.3rd.fa

Step 2

Bridge the contigs into scaffolds using Hi-C paired-end reads using SALSA. Based on the contigs we got in the previous step, we used the following command lines:

CONTIGFILE=F025.ctg.3rd.fa

HICFILE1=Hi-C pair-end-1.fastq

HICFILE2= Hi-C pair-end-2.fastq

PREFIX=F025arima

TEMPDIR=temp

threads=35

MAPQ\_FILTER=10

SALSADIR=SALSA\_DIR

FILTER=SALSA\_DIR/mapping\_pipeline/filter\_five\_end.pl

COMBINER= SALSA\_DIR /mapping\_pipeline/two\_read\_bam\_combiner.pl

bwa index $CONTIGFILE

bwa mem -t $threads $CONTIGFILE $HICFILE1 | samtools view -Sb - > aln-$PREFIX\\_1.bam

bwa mem -t $threads $CONTIGFILE $HICFILE2 | samtools view -Sb - > aln-$PREFIX\\_2.bam

samtools view -h aln-$PREFIX\\_1.bam | perl $FILTER | samtools view -Sb - > flt-$PREFIX\\_1.bam

samtools view -h aln-$PREFIX\\_2.bam | perl $FILTER | samtools view -Sb - > flt-$PREFIX\\_2.bam

samtools faidx $CONTIGFILE

perl $COMBINER flt-$PREFIX\\_1.bam flt-$PREFIX\\_2.bam samtools $MAPQ\_FILTER | samtools view -Sb -t $CONTIGFILE\.fai - | samtools sort -@ $threads-1 -o aln-$PREFIX\.bam -

bamToBed -i aln-$PREFIX\.bam > aln-$PREFIX\.bed

sort -k 4 --parallel=16 --temporary-directory=$TEMPDIR aln-$PREFIX\.bed > aln-$PREFIX\.srt.bed

python $SALSADIR/run\_pipeline.py -a $CONTIGFILE -l $CONTIGFILE\.fai -b aln-$PREFIX\.srt.bed -e AAGCTT -o $PREFIX -m yes -i 4 -s 1000000000 -c 500

Step 3

Fill gaps introduced from the scaffolding step using long reads using PBJelly.

Step 4

Polish the scaffolds using long reads using Racon. It is recommended to run Racon for 2-3 rounds. By running 3 rounds, we used the following command lines:

GENNAME=jelly.out.fasta

NANOPORE=nanopore long reads.fastq

PREFIX=F025

threads=48

types=map-ont

minimap2 -x $types -t $threads $GENNAME $NANOPORE > $PREFIX\.paf

racon -t $threads -u $NANOPORE $PREFIX\.paf $GENDIR/$GENNAME > racon.fasta

minimap2 -x $types -t $threads racon.fasta $NANOPORE > $PREFIX\2.paf

racon -t $threads -u $NANOPORE $PREFIX\2.paf racon.fasta > racon.2nd.fasta

minimap2 -x $types -t $threads racon.2nd.fasta $NANOPORE > $PREFIX\3.paf

racon -t $threads -u $NANOPORE $PREFIX\3.paf racon.2nd.fasta > racon.3rd.fasta

Step 5

Polish the scaffolds using short reads using Nextpolish. It is recommended to run Nextpolish for 2-3 rounds. By running 2 rounds, we used the following command lines:

nextpolish=/Nextpolish\_Dir/nextpolish1.py

input=racon.3rd.fasta

read1=Illumina paired-end-1.fastq

read2=Illumina paired-end-2.fastq

threads=48

round=2

for ((i=1; i<=${round};i++)); do

bwa index $input

bwa mem -t $threads $input $read1 $read2|samtools view --threads 3 -F 0x4 -b -|samtools fixmate -m --threads 3 - -|samtools sort -m 2g --threads 5 -|samtools markdup --threads 5 -r - sgs.sort.bam

samtools index -@ $threads sgs.sort.bam

samtools faidx $input

python $nextpolish -g $input -t 1 -p $threads -s sgs.sort.bam > genome.polishtemp.fa

input=genome.polishtemp.fa

bwa index $input

bwa mem -t $threads $input $read1 $read2|samtools view --threads 3 -F 0x4 -b -|samtools fixmate -m --threads 3 - -|samtools sort -m 2g --threads 5 -|samtools markdup --threads 5 -r - sgs.sort.bam

samtools index -@ $threads sgs.sort.bam

samtools faidx $input

python $nextpolish -g $input -t 2 -p $threads -s sgs.sort.bam > genome.nextpolish.fa

input=genome.nextpolish.fa

done;

Output

* The final assembly is genome.nextpolish.fa. Since it contains many lower-case letters to represent the bases added from Nextpolish in the sequence and its sequence name contains space, you need to process the sequence into upper-case letters and remove the space of the sequence name. You can use the following command line to do the process:

genome=genome.nextpolish.fa

genome-pre.py $genome > my\_genome.fa

**Gene annotation**

Dependencies

* Splign
* Blast
* Python3
* Bowtie2
* Samtools
* Bedtools
* Trinity
* STAR
* GFF3toolkit
* GffRead
* Infernal

Preparation

* Reference CDS isoforms from near species
* RNA-seq short reads
* Illumina paired-end sequencing reads
* rRNA database
* NR database
* Rfam database

Step 1

Map the reference CDS isoforms to the target assembly using Splign. We used the following command lines:

reference\_cds=reference\_CDS.fa

genome=my\_genome.fa

mkdir fasta\_dir

cp $genome fasta\_dir

cp $reference\_cds fasta\_dir

splign -mklds fasta\_dir

cd fasta\_dir

makeblastdb -dbtype nucl -parse\_seqids -in reference\_CDS.fa

makeblastdb -dbtype nucl -parse\_seqids -in my\_genome.fa

compart -qdb reference\_CDS.fa -sdb my\_genome.fa > cdna.compartments

cd ..

splign -ldsdir fasta\_dir -comps ./fasta\_dir/cdna.compartments > splign.output.ref

Step 2

Map the Illumina paired-end sequencing reads to the genome to get the region not supported by the short reads using Bowtie2 allowing no-mismatch. We used the following command lines:

genome=my\_genome.fa

r1=Illumina paired-end-1.fastq

r2= Illumina paired-end-2.fastq

threads=48

bowtie2-build $genome chicken

bowtie2 -p $threads -x chicken -1 $r1 -2 $r2 --score-min L,0,0 | samtools view -Sb -@ $threads-1 | samtools sort -@ $threads-1 > out.bam

bedtools genomecov -ibam out.bam -bga > out.bed

awk '$4=="0"{print $0}' out.bed > notsupport.region

Step 3

Map the RNA-seq short reads to rRNA database using Bowtie2 to get the unaligned reads, which are cleaned reads. Assemble the cleaned reads into transcripts using STAR and Trinity genome-guided method. We used the following command lines:

rrna=rrna\_database.fa

left=RNA-seq paired-end-1.fastq

right=RNA-seq paired-end-2.fastq

bowtie2-build $rrna rrna\_data

bowtie2 -p 48 --very-sensitive-local -x rrna\_data -1 $left -2 $right --un-conc-gz paired\_unaligned.fq.gz --un-gz unpaired\_unaligned.fq.gz

genome=my\_genome.fa

left=paired\_unaligned.fq.1

right=paired\_unaligned.fq.2

PREFIX=F025

threads=32

mkdir star

module load star/2.7.0c

STAR --runThreadN $threads --runMode genomeGenerate --genomeDir ./star --genomeFastaFiles $genome

STAR --genomeDir ./star --runThreadN $threads --readFilesIn $left $right --outFileNamePrefix $PREFIX --outSAMtype BAM SortedByCoordinate --outBAMsortingThreadN $threads --limitBAMsortRAM 214748364800

module load trinity/2.8.5

RNAbam=$PREFIX\Aligned.sortedByCoord.out.bam

Trinity --output Trinity\_GG --genome\_guided\_bam $RNAbam --genome\_guided\_max\_intron 200000 --CPU $threads --max\_memory 350G --verbose

Step 4

Predict non-coding RNAs using Infernal against Rfam database. We used the following command lines:

Rfam\_path=Path of Rfam database

Genome=my\_genome.fa

esl-seqstat $Genome

cmscan --cpu 48 --tblout result.tbl $Rfam\_path/Rfam.cm $Genome > result\_final.cmscan

Particularly, step 1, step 2, step 3 and step 4 can be executed simultaneously if there are enough memory on your cluster.

Step 5

Map the transcripts obtained in step 3 to the target assembly using Splign. We used the following command lines:

genome=my\_genome.fa

rna=transcripts.fa

mkdir fasta\_dir

cp $genome fasta\_dir

cp $rna fasta\_dir

splign -mklds fasta\_dir

cd fasta\_dir

makeblastdb -dbtype nucl -parse\_seqids -in transcripts.fa

makeblastdb -dbtype nucl -parse\_seqids -in my\_genome.fa

compart -qdb transcripts.fa -sdb my\_genome.fa > rna.compartments

cd ..

splign -ldsdir fasta\_dir -comps ./fasta\_dir/rna.compartments -type est > splign.output.rna

Step 6

Get the annotation results. We used the following command line:

cat parameter.txt annotation.pip > final.pip

chmod 711 final.pip

final.pip

You need to copy the parameter.txt from examples to your work directory and revise it to indict the path of your genome, reference CDS isoforms and their corresponding genes’ name, Splign output from reference CDS, Splign output from RNA-seq data, bed file of the genome region not supported by Illumina paired-end sequencing reads, non-coding RNA prediction result, path of NR database, minimum open reading frame length of RNA-unique genes (we recommend 300bp), minimum score of Splign output from RNA-seq data (we recommend 0.985 when RNA-seq data are from the same species, and you can use a smaller number if the RNA-seq data are not from the same species), number of iterations of Psi-blast, name of the output of Psi-blast, e-value of Psi-blast and number of threads to use of Psi-blast. Examples gives the examples of parameter.txt and the other files you need.

Output

* final\_annotation.gff3: final annotation results.

**Citation**