Finding genes and repeats in Rabiosa unphased haploid genome assembly

After making the genome assembly, the next step is to find genes and repeats in the assembly.

For finding genes, on the one hand, transcript and protein sequences from closely related genomes can be mapped to the assembly to help predict the position and structure of genes and on the other hand, ab initio or de novo gene prediction methods can be used to detect the open reading frame of genes in the assembly. The results of both approaches could be integrated to build final consensus gene models using a tool called EvidenceModeler.

For repeat annotation, known repeat sequences could be used to search for similar repeat sequences in the assembly. Those known repeat sequences could be obtained from different databases, such as Dfam and Repbase or one can build a de novo repeat library with the genome assembly using a tool called RepeatModeler.

The focus of this blog is de novo repeat library construction using RepeatModeler and gene prediction using EvidenceModeler.

This blog is too long, so just leave the conclusion here: using EvidenceModeler, **74883 high confidence genes were discovered, which account for 91% completeness of the proteome of Rabiosa as indicated by BUSCO analysis. High confidence genes, plus low confidence genes make up 94% completeness of the proteome.**

De novo repeat prediction using RepeatModeler and repeat masking using RepeatMasker

First, de novo repeat prediction using RepeatModeler2

RepeatModeler2 was installed via conda and before running its command, I need to export NINJA_DIR on our machine. Codes are following:

```
#!/bin/bash

export NINJA_DIR=/home/yutachen/public/Yutangchen/NINJA-0.98-cluster_only/NINJA

BuildDatabase -name Rabiosa rabiosa/211209_Rabiosa_new_pseudomolecules_v1+unanchored_contigs.fasta
RepeatModeler -database Rabiosa -pa 60 -LTRStruct >& rabiosa_repeatmodeler.out
```

The output of RepeatModerler2 is a file called Rabiosa-families.fa, which is consensus repeat sequences for every repeat sequence family found in Rabiosa assembly.

Since some predicted repeat sequences in the Rabiosa-families.fa might contain protein-coding sequences, these protein-coding sequences should be excluded. To do this, a tool called ProtExcluder.pl was used. Codes are following:

```
#!/bin/bash

# clean the de novo repeat library following the tutorial found here: https://www.biostars.org/p/411101/

makeblastdb -in genomic_resources_for_annotation/protein_for_annotation_renamed.fasta -title PA -dbtype prot
blastx -query /scratch/yutang/repeat/Rabiosa-families.fa \
    -db genomic_resources_for_annotation/protein_for_annotation_renamed.fasta \
    -out rabiosa_vs_protein.blastx -num_threads 60 -evalue 1e-6

/home/yutachen/public/Yutangchen/ProtExcluder1.2/ProtExcluder.pl -f 100 rabiosa_vs_protein.blastx /scratch/yutang/repeat/Rabiosa-families.fa
```

The idea of this tool is first blast the repeat sequence against some protein sequences using blastx and then exclude any sequences that are having a good blast hit in the repeat. The -f 100 in the command allows you to exclude 100 bp more at both ends of the blast hit. The protein_for_annotation_renamed.fasta is a file containing protein sequences from Brachypodium_distachyon_v3.0.pep.all.fa (Brachypodium), Hv_Morex.pgsb.Jul2020.aa.fa (Barley), Kyuss_1697_KYUS_proteins.fa (Kyuss, Lolium perenne), Lolium_2.6.1_V3_all_transcripts_PROT.fasta (Lolium_261, Lolium perenne), PepsiCo_OT3098_v2_predicted_genes_protein_seq_wrapped.fa (Hexaploid Oat), Secale_cereale_Lo7_2018v1p1p1.pgsb.Feb2019.all.aa.fasta (Rye).

After excluding potential protein-coding sequences using ProtExcluder.pl, a final repeat library file was obtained, Rabiosa_repeatmodeler_library.fasta

Second, annotating repeats in Rabiosa assembly with the de novo repeat library using RepeatMasker

RepeatMasker was installed via conda too and codes of RepeatMasker I used are following:

```
#!/bin/bash

RepeatMasker -pa 60 -xsmall -lib Rabiosa_repeatmodeler_library.fasta -no_is -gff -qq -norna -dir . Rabiosa.fasta
```

```
-xsmall means soft mask repetitive sequences, -no_is skips bacterial insertion element check, -gff creates an
additional General Feature Finding format output,
-qq very quick search for repeats with 10% less sensitivity, -norna does not mask RNA that are similar to SINE.
Check this https://www.animalgenome.org/bioinfo/resources/manuals/RepeatMasker.html for more options of
RepeatMasker.
```

The repeat feature is stored in a gff file called Rabiosa.fasta.out.gff. It would be interesting to see how repeats are distributed in the genome and also to see the composition of repeats.

Gene prediction for Rabiosa unphased assembly using EvidenceModeler

For this part, I adopted the genome annotation pipeline described in Martin Mascher's tritex barley genome paper with some minor changes based on my understanding of Lolium multiflorum genome, https://genomebiology.biomedcentral.com/articles/10.1186/s13059-019-1899-5

Map protein sequences from closely related geomes to Rabiosa using GenomeThreader

GenomeThreader can map protein sequences to a reference genome in a splice-aware manner and predict gene structure based on the mapping results. Command I used are following:

```
#!/bin/bash

# try to do genomethreader in parallel

# split protein fasta to chunkcs
gt splitfasta -numfiles 60 external_protein_nodot.fasta

# rename chunks
for x in $(seq 1 60)
do
   mv external_protein_nodot.fasta.${x} external_protein_${x}.fasta
done

# use unmasked reference
for x in $(seq 1 60)
do
   cp rabiosa.fasta protein_${x}/Rabiosa.fasta
  echo $x
done

seq 1 60 | parallel -j 5 -k 'startAlign.pl --genome=/scratch/yutang/gth_new/protein_{}/Rabiosa.fasta --dir=/scratch/yutang/gth_new
/protein_{} --prot=/scratch/yutang/gth_new/protein_{}/external_protein_{}.fasta --prg=gth --CPU=12 --args="-startcodon -finalstopcodon -
species rice -prseedlength 10 -prhdist 2 -gcmaxgapwidth 20000"'
```

GenomeThreader is slow, it is suggested to split the protein file to small chunks or split the reference by chromosomes, scaffolds or contigs. Here I did both, I split protein files to 60 chunks and use startAlign.pl from braker2 to map proteins to scaffolds separately in parallel. Check the FAQ of GenomeThreader here, https://genomethreader.org/faq.html. **-gcmaxgapwidth,** this option sets the maximum intron length for intron-aware mapping, here I used 20 Kb and I found lowering this value can largely reduces the usage of memory and significantly improves the speed. The **external_protein_nodot. fasta** is a copy of protein_for_annotation_renamed.fasta aforementioned. There are some dots in some protein sequences and these dots were removed by using sed substitution and the resulting file is **external_protein_nodot.fasta. gt** is a very handy tool set for processing fasta and gff files, check here for more information, http://genometools.org/tools.html

Aligning 60 protein files resulted in 60 gff files and these files were sorted and concatenated using GenomeTools. The final combined gff file was converted to the alignment gff format required by EvidenceModeler using a python script provided by Elisabeth. Codes used are following:

```
#!/bin/bash

# sort and merge gff files using genome tools

# first sort gff files
cp all the gff files to a new folder
```

```
mkdir mygff
for x in $(seq 1 60)
do
   cp protein_${x}/align_gth/gth.concat.aln mygff/external_protein_${x}.gff
   echo $x
done

# now sort every gff file
mkdir sortgff
for x in $(seq 1 60)
do
   gt gff3 -sort yes -force yes -tidy yes -o sortgff/sorted_${x}.gff mygff/external_protein_${x}.gff

   echo $x
done

# now merge gff files
mkdir mergegff
gt merge -force yes -tidy yes -o mergegff/merged.gff sortgff/*.gff

#now convert gff to alignment gff required by EVM

./prepare_gff.py merged.gff gth
```

This produced a gff file called merged.evm.gff

Meanwhile, I also downloaded reviewed protein sequences of poeae (uniprot_poeae.fa) from uniprot and mapped these protein sequences to Rabiosa using GenomeThreader to predict genes. Codes I used are following:

```
#!/bin/bash

gth -gff3out -skipalignmentout -genomic Rabiosa.fasta -protein uniprot_poeae.fa -species rice -paralogs -o gth_poeae.gff -prseedlength
7 -prhdist 4 -gcmaxgapwidth 20000 -startcodon -finalstopcodon -gcmincoverage 70
```

Then the resulting gth_poeae.gff file was converted to a gene prediction gff format required by EvidenceModeler. Codes used are following:

```
#!/bin/bash

$EVM_HOME/EvmUtils/misc/genomeThreader_to_evm_gff3.pl gth_poeae.gff > gth_poeae_evm.gff
```

Map RNA-Seq data of Rabiosa and transcripts from other genomes to Rabiosa assembly

We have some RNA-Seq data for Rabiosa, so I decided to map these RNA-Seq data to Rabiosa assembly to find transcripts. I first did quality check for these data, then removed adapters from the raw reads, next mapped the clean reads to assembly using Hisat2 and finally assembled transcripts using Stringtie2. Commands for Hisat2 and stringtie2 were learnt from this protocol, https://www.nature.com/articles/nprot.2016.095. Codes for quality check and adapter trimming I used are following:

```
#!/bin/bash

# trim the raw RNA seq reads, remove adapters and trim the left 10 biased bps

# first make a name list of all the fastq files
ls NextSeq500_20180307_NS163_o4112_DataDelivery/2018* | cut -f2 -d'/' | cut -f2 -d'.' | cut -f2 -d'_' > tissues.txt
ls NextSeq500_20180307_NS163_o4112_DataDelivery/2018* | cut -f2 -d'/' | cut -f2 -d'.' | cut -f3 -d'_' > replicate.txt
paste tissues.txt replicate.txt -d'_' | uniq > fastq.names
rm tissues.txt replicate.txt

# make a new folder to store trimmed reads
mkdir yutang_trimmomatic

# start trimming with trimmomatic
cat fastq.names | parallel -j 12 -k "trimmomatic PE -validatePairs -threads 5 -phred33 NextSeq500_20180307_NS163_o4112_DataDelivery
/20180307.A-New_{}_R1.fastq.gz NextSeq500_20180307_NS163_o4112_DataDelivery/20180307.A-New_{}_R2.fastq.gz -baseout
yutang_trimmomatic/{}_trimmed.fq.gz ILLUMINACLIP:True_seq_adapter.fa:2:30:10 HEADCROP:10 MINLEN:60"

# do fastqc with trimmed reads
mkdir yutang_trimmomatic_fastqc
cat fastq.names | parallel -j 12 -k "fastqc -t 5 -o yutang_trimmomatic_fastqc yutang_trimmomatic/{}_trimmed_1P.fq.gz
yutang_trimmomatic/{}_trimmed_2P.fq.gz"

# trimmomatic can remove adaptors but there are still some poly G or poly X sequencing remaining in the reads
# try fastp to trim the raw reads

# mkdir yutang_fastp
cat fastq.names | parallel -j 12 -k "fastp -i NextSeq500_20180307_NS163_o4112_DataDelivery/20180307.A-New_{}_R1.fastq.gz -I
NextSeq500_20180307_NS163_o4112_DataDelivery/20180307.A-New_{}_R2.fastq.gz -o yutang_fastp/{}_trimmed_R1.fq.gz -O
yutang_fastp/{}_trimmed_R2.fq.gz -g -y -x -f 10 -F 10 -w 5 -Q -l 40 -5 -3"
```

```
# do fastqc with trimmed reads
mkdir yutang_fastp_fastqc
cat fastq.names | parallel -j 12 -k "fastqc -t 5 -o yutang_fastp_fastqc yutang_fastp/{}_trimmed_R1.fq.gz yutang_fastp/{}_trimmed_R2.fq.
gz"
```

Using fastp instead of trimmomatic for adapter trimming as trimmomatic cannot remove adapters completely.

Codes for Hisat2 and Stringtie2:

```
#!/bin/bash

# use hisat2 to map RNA seq reads to rabiosa

# first build index for the genome
hisat2-build rabiosa/Rabiosa_211209.fa rabiosa/Rabiosa_211209

# now map RNA seq reads to rabiosa
mkdir yutang_hisat2
cat fastq.names | parallel -j 12 -k "hisat2 -p 5 --rna-strandness RF --max-intronlen 20000 --summary-file yutang_hisat2/{}.summary --dta -
x rabiosa/Rabiosa_211209 -1 yutang_fastp/{}_trimmed_R1.fq.gz -2 yutang_fastp/{}_trimmed_R2.fq.gz -S yutang_hisat2/{}.sam"

# sort sam
cat fastq.names | parallel -j 12 -k "samtools sort -@ 5 -o yutang_hisat2/{}.bam yutang_hisat2/{}.sam"

# assemble transcripts by stringtie
cat fastq.names | parallel -j 12 -k "stringtie -p 5 --rf -o yutang_hisat2/{}.gtf -l {} yutang_hisat2/{}.bam"

# merge transcripts
mkdir yutang_stringtie2
ls yutang_hisat2/*.gtf > mergelist.txt
stringtie --merge -p 20 -o yutang_stringtie2/stringtie_merged.gtf mergelist.txt

# index bam
cat fastq.names | parallel -j 12 -k "samtools index -@ 5 yutang_hisat2/{}.bam"

# convert gtf to gff3
gffread yutang_stringtie2/stringtie_merged.gtf -o yutang_stringtie2/stringtie_merged.gff3
```

Need to point out, when mapping RNA-Seq reads to Rabiosa assembly, the max intron length was set to 20 Kb, just to be consistent with protein alignment

Mapping transcripts from other genomes to Rabiosa using GMAP

Some transcripts of Lolium multiflorum and perenne were downloaded from NCBI and others were collected from Kyuss, Rabiosa NRGene's assembly and Lolium_261. A list of websites where I downloaded transcripts:

https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE144460

https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE78738

https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE141654

https://datacommons.cyverse.org/browse/iplant/home/shared/commons_repo/curated/Copetti_Kyuss_assembly_annotation_March_2021

https://ryegrassgenome.ghpc.au.dk/

https://zenodo.org/record/832654#.Ynovtp0za70

First these transcripts were converted to single-line fast and headers were renamed as transcript_1 and so on. Codes I used are following:

```
#!/bin/bash

mkdir transcript_new
for x in $(ls transcript)
do
  echo $x
  seqtk seq -l0 transcript/$x > test.fa
  awk '/^>/ {print ">transcript_"++n; next} {print $0}' test.fa > transcript_new/$x
  rm test.fa
done
```

Second transcripts were mapped to Rabiosa using GMAP, codes used are following:

```
#!/bin/bash

# first build the database
gmap_build -D gmap_test -d test Rabiosa.fasta
```

```
# map transcript I collected online to rabiosa
transcript=/home/yutachen/public/Yutangchen/Rabiosa_annotation/genomic_resources_for_annotation/transcript_new

for x in $(ls $transcript)
do
   echo "start mapping $x"
   gmap -D gmap_test -d test -t 60 -f 2 --max-intronlength-middle=20000 --split-large-introns --no-chimeras --min-identity=0.95 $transcript
/$x > $x.gff
   echo "$x mapping finished"
done
```

So, after mapping transcripts using GMAP, now gtf or gff files were compared using cuffcompare and then merged using Stringtie merge to remove any duplicated transcripts. After that, coding sequences were predicted using transdecoder in transcripts. Codes used are following:

```
#!/bin/bash

# compare gffs using cuffcompare
cuffcompare -s /home/yutachen/public/Yutangchen/Rabiosa_annotation/EVM/Rabiosa.fasta -G -i gff.list

# merge gffs using stringtie merge
stringtie --merge -p 20 -m 150 -o transcript_merged.gtf cuffcmp.combined.gtf

# find coding regions within transcripts using TransDecoder

# first step, construct transcript fasta file
gtf_genome_to_cdna_fasta.pl transcript_merged.gtf ../Rabiosa.fasta > transcripts.fasta

# convert gtf to alignment-gff3
gtf_to_alignment_gff3.pl transcript_merged.gtf > transcripts.gff3

# run transdecoder
TransDecoder.LongOrfs -t transcripts.fasta
TransDecoder.Predict -t transcripts.fasta

# generate a genome-based coding region annotation file:
cdna_alignment_orf_to_genome_orf.pl transcripts.fasta.transdecoder.gff3 transcripts.gff3 transcripts.fasta > transcripts.fasta.
transdecoder.genome.gff3
```

Below is what in the gff.list file:

```
/home/yutachen/public/Yutangchen/Rabiosa_annotation/EVM/stringtie/stringtie_merged.gff3
/home/yutachen/public/Yutangchen/Rabiosa_annotation/EVM/gmap/loliumMultiflorumCanonicalTranscripts.fa.gff
/home/yutachen/public/Yutangchen/Rabiosa_annotation/EVM/gmap/loliumPerenneCanonicalTranscripts.fa.gff
/home/yutachen/public/Yutangchen/Rabiosa_annotation/EVM/gmap/Rabiosa_v1_maker_transcripts_Lmu01_slp.fa.gff
/home/yutachen/public/Yutangchen/Rabiosa_annotation/EVM/gmap/COMPRE_31OCT2013.assemblies.fasta.gff
/home/yutachen/public/Yutangchen/Rabiosa_annotation/EVM/gmap/GSE78738_All-Unigene.fa.gff
/home/yutachen/public/Yutangchen/Rabiosa_annotation/EVM/gmap/GSE141654_L251_abiotic_Unigene-perennial_ryegrass.fa.gff
/home/yutachen/public/Yutangchen/Rabiosa_annotation/EVM/gmap/GSE144460_transcript.fa.gff
```

The transcripts.fasta.transdecoder.genome.gff3 file is later used for gene prediction with EvidenceModeler.

Ab initio gene prediction using Braker2 pipeline

Braker2 is a nice pipeline which uses ab initio gene prediction tools like Augustus to predict genes. Braker2 provides different recipes for ab initio gene prediction, for more detail, please check their github page, https://github.com/Gaius-Augustus/BRAKER

Since Augustus needs some hints to train its prediction model, so I used protein sequences of Kyuss as hints. Codes used are following:

```
#!/bin/bash

# source ~/anaconda3/etc/profile.d/conda.sh

# conda activate braker2

braker.pl --species=rabiosa --genome=rabiosa.fasta \
        --prot_seq=Kyuss_protein.fasta --prg=gth \
        --cores 48 --gff3 \
        --trainFromGth
```

This resulted in a file called augustus.hints.gff3, which contains gene models predicted by Augustus and this file is later used for building gene models with EvidenceModeler.

Build consensus gene models with evidence from protein alignment, transcripts alignment and ab initio gene prediction using EvidenceModeler

Simply, EvidenceModeler integrates evidence from different sources to build a consensus gene model for a locus, for more detail, please see this page, https://evidencemodeler.github.io/

Evidence that I collected for EvidenceModeler includes:

1. protein alignment using GenomeThreader with protein sequences from closely related genomes, **merged.evm.gff**
2. gene models produced by aligning uniprot_poeae.fa to Rabiosa assembly using GenomeThreader, **gth_poeae_evm.gff**
3. coding sequences predicted from transcript alignments, **transcripts.fasta.transdecoder.genome.gff3**
4. ab initio gene prediction produced by Braker2 pipeline, **augustus.hints.gff3**

These gff files were fed into EvidenceModeler to build consensus gene models, codes used are following:

```
#!/bin/bash

# use EVM to call sonsensus gene models with evidence from ab inito, transcript assembly and protein alignment

# make gene_predictions.gff3
cat augustus.hints.gff3 transcripts.fasta.transdecoder.genome.gff3 gth_poeae_evm.gff > gene_predictions.gff3

# Partitioning the Inputs
$EVM_HOME/EvmUtils/partition_EVM_inputs.pl --genome Rabiosa.fasta \
    --gene_predictions gene_predictions.gff3 --protein_alignments merged.evm.gff \
    --segmentSize 1000000 --overlapSize 100000 --partition_listing partitions_list.out --repeats Rabiosa.repeat.gff

# Generating the EVM Command Set
$EVM_HOME/EvmUtils/write_EVM_commands.pl --genome Rabiosa.fasta --weights `pwd`/weight.txt \
     --gene_predictions gene_predictions.gff3 --protein_alignments merged.evm.gff \
     --output_file_name evm.out  --partitions partitions_list.out > commands.list

# run EVM in parallel
cat commands.list | parallel -j 60 "{}"

# Combining the Partitions
$EVM_HOME/EvmUtils/recombine_EVM_partial_outputs.pl --partitions partitions_list.out --output_file_name evm.out

# convert evm.out to gff3
$EVM_HOME/EvmUtils/convert_EVM_outputs_to_GFF3.pl  --partitions partitions_list.out --output evm.out --genome Rabiosa.fasta

# combine all gff
find . -regex ".*evm.out.gff3" -exec cat {} \; > EVM.all.gff3

# extract protein
$EVM_HOME/EvmUtils/gff3_file_to_proteins.pl EVM.all.gff3 Rabiosa.fasta prot > EVM.prot.fasta

# extract cds
$EVM_HOME/EvmUtils/gff3_file_to_proteins.pl EVM.all.gff3 Rabiosa.fasta CDS > EVM.cds.fasta
```

Here is the content of the weight file:

```
ABINITIO_PREDICTION    AUGUSTUS    2
OTHER_PREDICTION    genomeThreader    8
OTHER_PREDICTION    transdecoder    5
PROTEIN    gth 3
```

After running EvidenceModeler, the completeness of the proteome was assessed by BUSCO version 5 with protein mode:

```
mkdir busco_prot_test
busco -i EVM.prot.fasta \
     -l /home/yutachen/public/Yutangchen/data_from_DLF/Columbus/busco_dir/viridiplantae_odb10 \
     -m proteins \
     -o busco_prot_test \
     -f \
     -c 20 \
     --offline
```

BUSCO results are: C:94.2%[S:81.2%,D:13.0%],F:1.9%,M:3.9%,n:430, This is very close to the genome assembly completeness, which means the genome annotation is as complete as the genome assembly. Very good!

CoreGF (core gene family) score is 88%, which is also very good. Codes used for coreGF analysis are following:

```
#!/bin/bash

# checking completeness of gene annotation using coreGF 2.5

myplaza=/home/yutachen/public/Yutangchen/Rabiosa_annotation/plaza5
```

```
# first balst EVM proteins against coreGF 2.5 proteome
diamond makedb --in $myplaza/PLAZA_2.5_proteome.fasta -d $myplaza/PLAZA_2.5_proteome --masking 0
diamond blastp -d $myplaza/PLAZA_2.5_proteome -q EVM/EVM.prot.fasta -o plaza/plaza_25_blastp.txt -f 6 --sensitive --masking 0

# calculate plaza coreGF score
python3 $myplaza/coreGF_plaza2.5_geneset.py $myplaza/coreGF_plaza2.5_monocots.txt plaza/plaza_25_blastp.txt > plaza
/plaza_25_blastp_coreGFscore.txt
```

Although EvidenceModeler output very good results, there are in total 326200 gene models, which is too many. This is because I didn't mask the repetitive regions in the assembly when running EvidenceModeler. There must be some TE genes in our candidate genes and they must be removed. Following the idea of confidence classification for the candidate gene set described in the tritex barley paper, I did following:

```
#!/bin/bash

# filter protein sequences after EVM
# follow the genome annotation pipeline described in Martin's tritex paper

# first use PTREP, a database of hypothetical proteins that contains deduced amino acid sequences in which, in many cases,
frameshifts have been removed,
# which is useful for the identification of divergent TEs having no significant similarity at the DNA level

# so use diamond to do blastp between EVM.prot.fasta and trep-db_proteins_Rel-19.fasta
trep=/home/yutachen/public/Yutangchen/Rabiosa_annotation/genomic_resources_for_annotation/repeat

# copy EVM.prot.fasta to folder filter

# make database for trep
diamond makedb --in $trep/trep-db_proteins_Rel-19.fasta -d $trep/trep-db_proteins_Rel-19 --masking 0
diamond blastp -d $trep/trep-db_proteins_Rel-19 -q filter/EVM.prot.fasta -o filter/EVM_prot_trep_blastp.txt -f 6 --sensitive --masking 0 -p
60

# align protein to uniprot_reviewd proteins
uniprot=/home/yutachen/public/Yutangchen/Rabiosa_annotation/genomic_resources_for_annotation/uniprot_reviewed
diamond makedb --in $uniprot/uniprot-reviewed_yes.fasta -d $uniprot/uniprot-reviewed_yes --masking 0
diamond blastp -d $uniprot/uniprot-reviewed_yes -q filter/EVM.prot.fasta -o filter/EVM_prot_uniprot_blastp.txt -f 6 --sensitive --masking
0 -p 60

# search for domain with pfam
mypfam=/home/yutachen/public/Yutangchen/Rabiosa_annotation/genomic_resources_for_annotation/Pfam
hmmsearch --cpu 60 --domtblout filter/EVM_pfam.domtblout $mypfam/Pfam-A.hmm filter/EVM.prot.fasta
```

In addition, following what was described in Kyuss paper by Elisabeth, cds of candidate gene models was blast against annotated repetitive regions to check if there is a large overlap between gene model and repeat, here's what I did:

```
#!/bin/bash

# blast EVM.cds.fasta against annotated repeats in rabiosa

# first extract annotated repeats from rabiosa
bedtools getfasta -fi Rabiosa.fasta -bed repeatmasker/Rabiosa.repeat.gff > repeatmasker/Rabiosa_repeat.fasta

# build blastn database
makeblastdb -in repeatmasker/Rabiosa_repeat.fasta -dbtype nucl
blastn -query filter/EVM.cds.fasta -db repeatmasker/Rabiosa_repeat.fasta -outfmt 6 -evalue 1e-10 -num_threads 48 -max_target_seqs 5
> filter/EVM_repeat_blastn.txt
```

Finally, I wrote a R script to parse each blast results and assigned gene models to high confidence set, low confidence set and repeat set, here's my R script:

```
# assign gene model to high and low confidence set

# set working directory
setwd("P:/Yutangchen/Rabiosa_annotation/EVM/filter")

library(stringr)

# import data
protfai <- read.table("EVM.prot.fasta.fai", header = F, stringsAsFactors = F)
cdsfai <- read.table("EVM.cds.fasta.fai", header = F, stringsAsFactors = F)
uniprot <- read.table("EVM_prot_uniprot_blastp.txt", header = F, stringsAsFactors = F)
pfam <- read.table("EVM_pfam.domtblout", header = F, stringsAsFactors = F)
trep <- read.table("EVM_prot_trep_blastp.txt", header = F, stringsAsFactors = F)
repeatmask <- read.table("EVM_repeat_blastn.txt", header = F, stringsAsFactors = F)
```

```
# sort the gene ID numerically
evmprot <- type.convert(as.data.frame(str_split_fixed(protfai$V1, "[.]", 4)), as.is = T)
str(evmprot)
evmprot <- evmprot[order(evmprot$V3, evmprot$V4), ]
evmprot$ID <- paste(evmprot$V1, evmprot$V2, evmprot$V3, evmprot$V4, sep = ".")
evmprot <- evmprot[, 3:5]
names(evmprot) <- c("CHR", "Num", "ID")

# add protein and cds length to evmprot
evmprot$ProtL <- protfai[order(match(protfai$V1, evmprot$ID)), 2]
evmprot$CdsL <- cdsfai[order(match(cdsfai$V1, evmprot$ID)), 2]
rm(cdsfai, protfai)

# now add protein alignment evidence or repeat alignment evidence to evmprot

# uniprot
uniprot_best_hit <- uniprot[!duplicated(uniprot$V1), ]
uniprot_best_hit <- uniprot_best_hit[uniprot_best_hit$V11 <= 1e-10, ]
write.table(uniprot_best_hit, "uniprot_best_hit.txt", quote = F, sep = "\t", row.names = F, col.names = F)

evmprot$Uniprot <- evmprot$ID %in% uniprot_best_hit$V1
evmprot$Uniprot[evmprot$Uniprot == "False"] <- 0
evmprot$Uniprot[evmprot$Uniprot == "True"] <- 1

rm(uniprot, uniprot_best_hit)

# pfam
pfam_best_hits <- pfam[pfam$V22 >= 0.90, ]
evmprot$Pfam <- evmprot$ID %in% pfam_best_hits$V1
evmprot$Pfam[evmprot$Pfam == "False"] <- 0
evmprot$Pfam[evmprot$Pfam == "True"] <- 1

rm(pfam, pfam_best_hits)

# trep
trep_best_hit <- trep[!duplicated(trep$V1), ]
trep_best_hit <- trep_best_hit[trep_best_hit$V11 <= 1e-10, ]
xx <- evmprot[evmprot$ID %in% trep_best_hit$V1, ]
xx <- xx[order(match(xx$ID, trep_best_hit$V1)), ]
trep_best_hit$overlap <- (trep_best_hit$V8 - trep_best_hit$V7)/xx$ProtL
trep_best_hit <- trep_best_hit[trep_best_hit$overlap >= 0.75, ]
evmprot$Trep <- evmprot$ID %in% trep_best_hit$V1
evmprot$Trep[evmprot$Trep == "False"] <- 0
evmprot$Trep[evmprot$Trep == "True"] <- 1
rm(trep, trep_best_hit, xx)

# repeatmask
repeatmask_best_hit <- repeatmask[!duplicated(repeatmask$V1), ]
repeatmask_best_hit <- repeatmask_best_hit[repeatmask_best_hit$V11 <= 1e-10, ]
xx <- evmprot[evmprot$ID %in% repeatmask_best_hit$V1, ]
xx <- xx[order(match(xx$ID, repeatmask_best_hit$V1)), ]
repeatmask_best_hit$overlap <- (repeatmask_best_hit$V8 - repeatmask_best_hit$V7)/xx$CdsL
repeatmask_best_hit <- repeatmask_best_hit[repeatmask_best_hit$overlap >= 0.75, ]
evmprot$repeatmask <- evmprot$ID %in% repeatmask_best_hit$V1
evmprot$repeatmask[evmprot$repeatmask == "False"] <- 0
evmprot$repeatmask[evmprot$repeatmask == "True"] <- 1
rm(repeatmask, repeatmask_best_hit, xx)

# now assign gene models to different sets with different level of confidence
EVM_hc <- evmprot[(evmprot$Uniprot == 1 | evmprot$Pfam == 1) & evmprot$Trep == 0 & evmprot $repeatmask == 0, ]
EVM_lc <- evmprot[evmprot$Uniprot == 0 & evmprot$Pfam == 0 & evmprot$Trep == 0 & evmprot $repeatmask == 0, ]
EVM_rep <- evmprot[evmprot$Trep == 1 | evmprot $repeatmask == 1, ]

write.table(EVM_hc, "EVM_high_confidence_gene_model.txt", quote = F, sep = "\t", row.names = F, col.names = F)
write.table(EVM_lc, "EVM_low_confidence_gene_model.txt", quote = F, sep = "\t", row.names = F, col.names = F)
write.table(EVM_rep, "EVM_high_repeat_gene_model.txt", quote = F, sep = "\t", row.names = F, col.names = F)
```

Extract protein sequence of each set:

```
cut -f3 EVM_high_confidence_gene_model.txt > EVM_hc.lst # 74883 genes
cut -f3 EVM_low_confidence_gene_model.txt > EVM_lc.lst # 123860 genes
cat EVM_hc.lst EVM_lc.lst > EVM_gene.lst # 198743 genes

seqtk subseq EVM.prot.fasta EVM_hc.lst > EVM.prot.hc.fasta
seqtk subseq EVM.prot.fasta EVM_lc.lst > EVM.prot.lc.fasta
seqtk subseq EVM.prot.fasta EVM_gene.lst > EVM.prot.gene.fasta
```

Did BUSCO again on EVM.prot.hc.fasta and EVM.prot.gene.fasta, results are following:

C:91.0%[S:78.4%,D:12.6%],F:1.9%,M:7.1%,n:430

C:94.2%[S:81.2%,D:13.0%],F:1.9%,M:3.9%,n:430

High confidence gene is defined as it has a "good" blast hit in Uniprot protein or in Pfam database but overlaps 25% or lower with a repeat. Low confidence gene is the one having no "good" blast hit in Uniprot and Pfam database and 25% lower overlap with a repeat. Repeat gene is the one having more than 75% overlap with a repeat regardless of having a good hit in Uniprot or Pfam database. However, the low confidence gene may also become a high confidence gene if in the future more information of the gene is reported. F or whom may use the gene annotation, please keep in mind, you should decide it is a high or low confidence gene for your study, not me!