Advanced repeat library for Maker2

Xabier Vázquez-Campos

2017-11-20

Contents

This protocol is heavily based on the *Repeat Library Construction-Advanced* from the Maker wiki and contributed by Ning Jiang, Megan Bowman, and Kevin Childs from Michigan State University.

Requisites

Programs and/or scripts

- MITE Hunter
- GenomeTools
- RepeatMasker (>=4.0.7)
- RepeatModeler (>=1.0.9)
- BLAST+
- MUSCLE
- BioPerl
- HMMER

- CRL scripts
- ProteinExcluder 1.2
- Anaconda, not strictly necessary

  NOTE: a usual installation of HMMER is required. ProteinExcluder depends on the `easl` miniapps from HMMER and neither Conda or Ubuntu installs them.

Modification of ProteinExcluder

ProteinExcluder 1.2 was modified to use `samtools faidx` instead of `esl-sfetch` due constant errors. The modifications were based on this entry from the maker-devel mail list.
If it works for you, you don't need to do anything. Otherwise, the modifications are made in the `mspesl-sfetch.pl` script located in the ProtExcluder folder:

```
# at line 17 change:
`/PATH/TO/HMMER/binaries/esl-sfetch --index $ARGV[0]`
# by
`samtools faidx $ARGV[0]`

# at line 26:
`/PATH/TO/HMMER/binaries/esl-sfetch -c $from..$to $ARGV[0] $line[7] >> $ARGV[3]`
# by
`samtools faidx $ARGV[0] $line[7]:$from-$to >> $ARGV[3]`
```

If you don't have `sammtools` in your system path, you will need to set the full path to its binary. Otherwise, modify as indicated.

Databases and libraries

- Eukaryotic tRNAs. Derived from tRNAscan-SE.
- Transposases
  - All transposase proteins
  - DNA transposases
- Modified UniProt-SwissProt without transposases, see below

Filter transposases from SwissProt

You can skip this if you already have a curated SwissProt database free of transposases

Search SwissProt matches within the Transposase protein database

```
blastp -query uniprot_sprot.fasta -db ${TpasesPROT} -evalue 1e-10 -max_hsps 1 -max_target_seqs 1 -num_t

cut -f 1 sprot_tpasesprot.tab > sprot_tpaseprot.txt

grep ">" uniprot_sprot.fasta | grep -v -f sprot_tpaseprot.txt | sed 's/^>//g' | sed 's/[ ].*//g' > spro

xargs samtools faidx uniprot_sprot.fasta < sprot_notpasesprot.txt > uniprot_sprot_notpasesprot.fasta
```

1. Searches the SwissProt transposases.

2. Gets the list of SwissProt proteins with matches.

3. Generate a list of SwissProt proteins to keep.

4. Generate a SwissProt-filtered fasta file.

Now time to do the same with the Transposase DNA database:

```
blastp -query uniprot_sprot_notpasesprot.fasta -db ${TpasesDNA} -evalue 1e-10 -max_hsps 1 -max_target_s

cut -f 1 sprot_tpasesdna.tab > sprot_tpasedna.txt

grep ">" uniprot_sprot_notpasesprot.fasta | grep -v -f sprot_tpasedna.txt | sed 's/^>//g' | sed 's/[ ].

xargs samtools faidx uniprot_sprot_notpasesprot.fasta < sprot_clean.txt > uniprot_sprot_clean.fasta
```

Databases to index

```
makeblastdb -in ${SPROT} -dbtype prot
makeblastdb -in ${EUK_tRNA} -dbtype nucl
makeblastdb -in ${TpasesDNA} -dbtype prot
makeblastdb -in ${TpasesPROT} -dbtype prot
```

Directory structure

```
# \
# |-- ${MYGENOME}
# |   |-- adv_repeats
# |       |-- LTR
# |       |-- MITE
```

Variables

Programs

```
DIR_MITE=/home/xabi/MITE_Hunter

DIR_CRL=/home/xabi/CRL_Scripts1.0

DIR_PE=/home/xabi/ProtExcluder1.2
```

Libraries and files

```
BASE_PATH=~/Desktop/SBI_projects/Chongmei/pugra
AR_PATH=${BASE_PATH}/adv_repeats
GENOME=/home/xabi/Desktop/SBI_projects/Chongmei/assembly/run_6_lcutoff_6k_lcutoffpr_6k/pilon_error_corr
PREFIX=pugra
CPU=4
EUK_tRNA=/home/xabi/Desktop/adv_rep_libs/eukaryotic-trnas.fa
TpasesDNA=/home/xabi/Desktop/adv_rep_libs/Tpases020812DNA
TpasesPROT=/home/xabi/Desktop/adv_rep_libs/Tpases020812
SPROT=/home/xabi/Desktop/adv_rep_libs/uniprot_sprot_clean.fasta
```

`INPUT` is the assembly/genome fasta file.
`PREFIX` is an identifier/prefix/index name. Choose something identificative for your genome.

Input genome

Many of the tools and scripts along this workflow don't handle well special characters in the fasta headers or long headers.
So, it is very recommendable to simplify those headers. While a simple `awk`-base substitution would do, using a script like `simplifyFastaHeaders.pl` allows to keep a mapping file with the correspondences between old and new headers:

```
perl ~/simplifyFastaHeaders.pl ${GENOME} ${PREFIX} ${GENOME%.fasta}.simp.fasta ${GENOME%.fasta}.map

INPUT=${GENOME%.fasta}.simp.fasta
```

MITEs (Miniature Inverted-repeat Transposable Elements)

```
cd ${AR_PATH}
mkdir -p MITE
cd MITE
${DIR_MITE}/MITE_Hunter_manager.pl -i ${INPUT} -g ${PREFIX} -n ${CPU} -S 12345678
cat ${PREFIX}_Step8_*.fa > MITE.lib
mv MITE.lib ../
cd ..
```

MITE Hunter creates a lot of intermediate files, e.g. I got 8.2 GB of files for a genome of 80+ Mbp (86 MB fasta file). They can be removed after finishing this protocol.

LTR (Long Terminal Repeat) retrotransposons

In this protocol, we distinguish evolutionary recent LTRs in which the terminal repeats have a minimum of 99% similarity, and evolutionary old LTRs, with a minimum of 85% similarity.

IMPORTANT: LTR harvest doesn't like certain special characters, including "." and "_", in the fasta headers. It also splits the headers at the spaces.

Recent LTRs (99%)

Find candidate elements

```
cd ${AR_PATH}
mkdir -p LTR
cd LTR

gt suffixerator -db ${INPUT} -indexname ${PREFIX} -tis -suf -lcp -des -ssp -dna

gt ltrharvest -index ${PREFIX} -out ${PREFIX}.out99 -outinner ${PREFIX}.outinner99 -gff3 ${PREFIX}.gff9
```

- `-minlenltr 100 -maxlenltr 6000`: the size of the terminal repeats between 100-6000 bp, and 99% identical `-similar 99`.
- `-mindistltr 1500 -maxdistltr 25000`: the size of the entire element between 1.5-25 kbp.
- `-motif tgca`: both terminal repeats must end with "TG...GA".

- Elements must be flanked by a target site duplication (TSD) of 5 bp `-maxtsd 5` and placed within 10 bp from the end of the elements `-vic 10`.

Find elements with PPT (poly purine tract) or PBS (primer binding site)

```
gt gff3 -sort ${PREFIX}.gff99 > ${PREFIX}.gff99.sort
gt ltrdigest -trnas ${EUK_tRNA} ${PREFIX}.gff99.sort ${PREFIX} > ${PREFIX}.gff99.dgt
perl ${DIR_CRL}/CRL_Step1.pl --gff ${PREFIX}.gff99.dgt
```

Additional filtering of the candidate elements

```
perl ${DIR_CRL}/CRL_Step2.pl --step1 CRL_Step1_Passed_Elements.txt --repeatfile ${PREFIX}.out99 --resul
mkdir -p fasta_files
mv Repeat_*.fasta fasta_files/
mv CRL_Step2_Passed_Elements.fasta fasta_files/
cd fasta_files/
perl ${DIR_CRL}/CRL_Step3.pl --directory ./ --step2 CRL_Step2_Passed_Elements.fasta --pidentity 60 --se
mv CRL_Step3_Passed_Elements.fasta ../
cd ..
```

Identify elements with nested insertions

```
perl ${DIR_CRL}/ltr_library.pl --resultfile ${PREFIX}.result99 --step3 CRL_Step3_Passed_Elements.fasta

cat lLTR_Only.lib ${AR_PATH}/MITE.lib > repeats_to_mask_LTR99.fasta
```

Search the repeats (so far) with RepeatMasker in Katana:

```
module purge
module load perl/5.20.1
module load repeatmasker/4.0.7

PREFIX=pugra
AR_PATH=${BASE}/${PREFIX}/adv_repeats
SIM_VAL=99
library=${AR_PATH}/LTR/repeats_to_mask_LTR${SIM_VAL}.fasta

cd ${AR_PATH}/LTR

${DIR_RM1}/RepeatMasker -pa ${PBS_NUM_PPN} -lib ${library} -nolow -dir ./ ${AR_PATH}/LTR/${PREFIX}.outi
```

Back to local:

```
perl ${DIR_CRL}/cleanRM.pl ${PREFIX}.outinner99.out ${PREFIX}.outinner99.masked > ${PREFIX}.outinner99.

perl ${DIR_CRL}/rmshortinner.pl ${PREFIX}.outinner99.unmasked 50 > ${PREFIX}.outinner99.clean

blastx -query ${PREFIX}.outinner99.clean -db ${TpasesDNA} -evalue 1e-10 -num_threads ${CPU} -num_descri

perl ${DIR_CRL}/outinner_blastx_parse.pl --blastx ${PREFIX}.outinner99.clean_blastx.out.txt --outinner
```

Building examplars

```
perl ${DIR_CRL}/CRL_Step4.pl --step3 CRL_Step3_Passed_Elements.fasta --resultfile ${PREFIX}.result99 --

makeblastdb -in lLTRs_Seq_For_BLAST.fasta -dbtype nucl

blastn -query lLTRs_Seq_For_BLAST.fasta -db lLTRs_Seq_For_BLAST.fasta -evalue 1e-10 -num_descriptions 1

makeblastdb -in Inner_Seq_For_BLAST.fasta -dbtype nucl

blastn -query Inner_Seq_For_BLAST.fasta -db Inner_Seq_For_BLAST.fasta -evalue 1e-10 -num_descriptions 1

perl ${DIR_CRL}/CRL_Step5.pl --LTR_blast lLTRs_Seq_For_BLAST.fasta.out --inner_blast Inner_Seq_For_BLAS
```

Old LTRs (85%)

Before proceed, remove stuff from the Recent LTRs step to avoid problems

```
rm fasta_files/* CRL_Step*
```

Find candidate elements (85%)

```
gt ltrharvest -index ${PREFIX} -out ${PREFIX}.out85 -outinner ${PREFIX}.outinner85 -gff3 ${PREFIX}.gff8
```

- Since the terminal sequence motif is not specified, only elements with terminal sequences with patterns that are previously reported are retained.
- We don't need to specify `-similar 85` as it is the default value.

Find elements with PPT or PBS (85%)

```
gt gff3 -sort ${PREFIX}.gff85 > ${PREFIX}.gff85.sort

gt ltrdigest -trnas ${EUK_tRNA} ${PREFIX}.gff85.sort ${PREFIX} > ${PREFIX}.gff85.dgt

perl ${DIR_CRL}/CRL_Step1.pl --gff ${PREFIX}.gff85.dgt
```

Additional filtering of the candidate elements (85%)

```
perl ${DIR_CRL}/CRL_Step2.pl --step1 CRL_Step1_Passed_Elements.txt --repeatfile ${PREFIX}.out85 --resul

mkdir -p fasta_files

mv Repeat_*.fasta fasta_files

mv CRL_Step2_Passed_Elements.fasta fasta_files

cd fasta_files

perl ${DIR_CRL}/CRL_Step3.pl --directory ./ --step2 CRL_Step2_Passed_Elements.fasta --pidentity 60 --se

mv CRL_Step3_Passed_Elements.fasta ..

cd ..
```

Identify elements with nested insertions (85%)

```
perl ${DIR_CRL}/ltr_library.pl --resultfile ${PREFIX}.result85 --step3 CRL_Step3_Passed_Elements.fasta

cat lLTR_Only.lib MITE/MITE.lib > repeats_to_mask_LTR85.fasta
```

Search the repeats (so far) with RepeatMasker in Katana:

```
DIR_RM1=/srv/scratch/z3382651/RepeatMasker
AR_PATH=/srv/scratch/z3382651/sbi/pugra/adv_repeats2
PREFIX=pugra
library=${AR_PATH}/LTR/repeats_to_mask_LTR85.fasta

cd ${AR_PATH}/LTR

${DIR_RM1}/RepeatMasker -pa ${PBS_NUM_PPN} -lib ${library} -nolow -dir . ${AR_PATH}/LTR/${PREFIX}.outinn
```

And, go back to local:

```
perl ${DIR_CRL}/cleanRM.pl ${PREFIX}.outinner85.out ${PREFIX}.outinner85.masked > ${PREFIX}.outinner85.

perl ${DIR_CRL}/rmshortinner.pl ${PREFIX}.outinner85.unmasked 50 > ${PREFIX}.outinner85.clean

blastx -query ${PREFIX}.outinner85.clean -db ${TpasesDNA} -evalue 1e-10 -num_threads ${CPU} -num_descrip

perl ${DIR_CRL}/outinner_blastx_parse.pl --blastx ${PREFIX}.outinner85.clean_blastx.out.txt --outinner
```

Building examplars (85%)

```
perl ${DIR_CRL}/CRL_Step4.pl --step3 CRL_Step3_Passed_Elements.fasta --resultfile ${PREFIX}.result85 --

makeblastdb -in lLTRs_Seq_For_BLAST.fasta -dbtype nucl

blastn -query lLTRs_Seq_For_BLAST.fasta -db lLTRs_Seq_For_BLAST.fasta -evalue 1e-10 -num_descriptions 1

makeblastdb -in Inner_Seq_For_BLAST.fasta -dbtype nucl

blastn -query Inner_Seq_For_BLAST.fasta -db Inner_Seq_For_BLAST.fasta -evalue 1e-10 -num_descriptions 1

perl ${DIR_CRL}/CRL_Step5.pl --LTR_blast lLTRs_Seq_For_BLAST.fasta.out --inner_blast Inner_Seq_For_BLAS
```

Consolidate LTRs

Because some of the LTR99 will be also contained in LTR85, we mask `LTR85.lib` with `LTR99.lib` to remove any redundant LTR in LTR85 that is already in LTR99:

```
DIR_RM1=/srv/scratch/z3382651/RepeatMasker
AR_PATH=/srv/scratch/z3382651/sbi/pugra/adv_repeats2
PREFIX=pugra
library=${AR_PATH}/LTR/LTR99.lib

cd ${AR_PATH}/LTR

${DIR_RM1}/RepeatMasker -pa ${PBS_NUM_PPN} -lib ${library} -dir . ${AR_PATH}/LTR/LTR85.lib
```

And back in local again, we now create the `FinalLTR85.lib` without the elements already present in `LTR99.lib` and merge them to create `allLTR.lib`, which contains evolutionary recent and distant LTR elements.

```
perl ${DIR_CRL}/remove_masked_sequence.pl --masked_elements LTR85.lib.masked --outfile FinalLTR85.lib

cat LTR99.lib FinalLTR85.lib > allLTR.lib
```

Repetitive elements with RepeatModeler

Merge MITE and LTR libraries:

```
cd ${ADV_REP}
cat LTR/allLTR.lib MITE/MITE.lib > allMITE_LTR.lib
```

Mask the genome:

```
DIR_RM1=/srv/scratch/z3382651/RepeatMasker
AR_PATH=/srv/scratch/z3382651/sbi/pugra/adv_repeats2
PREFIX=pugra
library=${AR_PATH}/allMITE_LTR.lib
INPUT=assembly.fasta

cd ${AR_PATH}/LTR

${DIR_RM1}/RepeatMasker -pa ${PBS_NUM_PPN} -lib ${library} -dir . ${INPUT}
```

Back to local. This removes the masked elements (no need to predict them again)

```
perl ${DIR_CRL}/rmaskedpart.pl ${INPUT##*/}.masked 50 > um_${INPUT##*/}
```

Now run RepeatModeler on Katana:

```
module purge
module load perl/5.20.1
module load recon/1.08
module load repeatscout/1.05
module load trf/4.09
module load rmblast/2.2.28
# module load rmblast/2.6.0
module load repeatmasker/4.0.7
module load repeatmodeler/1.0.10

INPUT=/srv/scratch/z3382651/sbi/pugra/assembly/run_6_lcutoff_6k_lcutoffpr_6k/pilon_error_correction/pil
PREFIX=pugra
BASE=/srv/scratch/z3382651/sbi/pugra/adv_repeats2

cd ${BASE}

BuildDatabase -name um_${INPUT##*/}db -engine ncbi um_${INPUT##*/}

nohup RepeatModeler -pa ${PBS_NUM_PPN} -database um_${INPUT##*/}db >& um_${PREFIX}.out
```

RepeatModeler is able to identify some repeats but not other. Let's separate them and keep processing the *unknowns*:

```
perl ${DIR_CRL}/repeatmodeler_parse.pl --fastafile consensi.fa.classified --unknowns repeatmodeler_unkn
```

`repeatmodeler_unknowns.fasta` are searched against the transposase database and the matching sequences are classified as such:

```
blastx -query repeatmodeler_unknowns.fasta -db ${TpasesPROT} -evalue 1e-10 -num_descriptions 10 -out mo

perl ${DIR_CRL}/transposon_blast_parse.pl --blastx modelerunknown_blast_results.txt --modelerunknown re
```

The completely unknown elements are renamed and all the identified ones (from RepeatModeler and Blast) merged:

```
mv unknown_elements.txt ModelerUnknown.lib
cat identified_elements.txt repeatmodeler_identities.fasta > ModelerID.lib
```

Excluding gene fragments

The last step involves evaluating if some of this *unknown* repeats are just fragments of genes mistakenly detected as repeats:

```
for lib in ModelerID.lib allLTR_rename.lib MITE.lib ModelerUnknown.lib
do
blastx -query ${lib} -db ${SPROT} -evalue 1e-10 -num_descriptions 10 -num_threads ${CPU} -out ${lib}_bla
${DIR_PE}/ProtExcluder.pl ${lib}_blast_results.txt ${lib}
echo -e "${lib}\tbefore\t$(grep -c ">" ${lib})\tafter\t$(grep -c ">" ${lib}noProtFinal)"
done
```

> The default options `-f 50` excludes 50 bp upstream and downstream of the blast hit, while remaining fragments shorter than that are completely removed.
>
> The final (wanted) output will be the `${lib}noProtFinal` files.

All filtered known repeats are merged:

```
cat MITE.libnoProtFinal allLTR_rename.libnoProtFinal ModelerID.libnoProtFinal > KnownRepeats.lib
```

And finally, we create the final repeat library:

```
cat KnownRepeats.lib ModelerUnknown.libnoProtFinal > allRepeats.lib
```