AUTOMITAUR v1.0.1
A tool for the *de novo* detection of MITEs

2013, February 2th

Copyright 2013 Aurélie Hua-Van, ahuavan@legs.cnrs-gif.fr

Please report any bugs, comments or questions to ahuavan@legs.cnrs-gif.fr

I- Installation

I-1 Companions programs

You must have installed on the computer:

- Python2.6 or later
- BioPython 1.58 or later
- Usearch5 (Robert Edgar) (usearch5.2 won't work)
  http://drive5.com/usearch/usearch5.1.html
- Muscle (Robert Edgar)
  http://www.drive5.com/muscle/downloads.htm
- Mafft version 6
  http://mafft.cbrc.jp/alignment/software/
- TRF (Tandem Repeat Finder)
  http://tandem.bu.edu/trf/trf.download.html
- Blast2.2.20 (or Blast+, recommended with AutoMITAur.v1.0.1- Blast2.23 may cause problems)
  ftp://ftp.ncbi.nlm.nih.gov/blast/executables/release/
- IRF (Inverted Repeat Finder) (IRF fails to run on 64-bits Unix computers)
  http://tandem.bu.edu/irf/irf.download.html
- Seqmagick (not needed anymore on AutoMITAur.v1.0.1)
  http://fhcrc.github.com/seqmagick/

Check that the locations of the programs are in your path, and that all programs are executable.

Rename (or create symbolic link in your path) the following programs that comes with very long name including version and platforms:

Irf404_xxxxxx must be renamed **irf**

Trf309-xxxx must be renamed **trf**

Usearch5xxxxxxx must be renamed **uclust**

Musclexxxxxxx must be renamed **muscle**

Command line: sudo ln –s ~/…./irf404_xxxxxx   ~/…./irf

I-2 automitaur scripts

Automitaur consist of 10 Python scripts (.py). The main one is automitaur.py

All scripts are in the directory AutoMITAur.v1.0.1 that can be installed anywhere.

Dezip AutoMITAur.v1.0.1.zip and place the directory where you want.

You will have to be in that directory to run automitaur script. The genome may be elsewhere and you have to provide the full path to genome when running automitaur. All files created by automitaur will be stored in the *genome.dir* directory created in the genome directory.

Open a terminal:

```
>cd Full/path/to/AutoMitaur.vx.x.x

>python automitaur.py –i Full/path/to/genome   (minimal command line)
```

II- Pipeline

Important output files

AutoMITAur generates lots of intermediate files that are kept until you run the script CleanAutomitaur.py

```
python CleanAutomitaur.py Full/Path/to/genome(M).dir (NOT YET FUNCTIONAL)
```

The script will remove all unnecessary files.

Output files that will be kept (in genome.dir/prog) (when functional)

Genome.xxxx.dat

Options : options and parameters used

Statistics: number of cluster and copies kept at each step, duration of the step

BilanAutonomous : cluster displaying potentially autonomous partner, and number

BilanTSD: best TSD retained for each family

BilanCopies.txt: coordinates of all copies

BilanTIR: position of TIR of all families

BilanTIR.fa: fasta file of TIR consensus sequences of all families

BilanNbSeq: number of total and independent copies for all families

FinalBilan: Main characteristics of all families (tab format)

All files in Cons

0- Genome input file and parameters

0-1 Genome file format

The input file (genome_file) must be in **fasta** format and may contain one or more sequences. It is highly recommended that the names of the sequences be short and do not contain spaces, or other unusual characters (including '|').

You can change the name of the sequence by typing :

```
python ChangeSequenceName.py Full/path/to/genome
```

You will be asked for the regular expression to keep, then for a prefix to add. A new genome file with N added at the end will be created.

Example:

>>>python ChangeSequenceName.py *Full/path/to/Genus_species.fasta*

>Genus species 1.0 blah blah blah

pattern to keep (re) ?: 1.\d+ (This will keep expression starting with 1. and ending with any numbers. Important number identifying the sequence)

item to add ? supercontig_

Result: a new genome file named *Genus_species.fastaN* with sequences renamed >supercontig_1.x

BEWARE: If the pattern to keep is not found in the name of a sequence, the sequence will be removed from the new file!

The input file does not need to be in the same directory as Automitaur, if you give the full path of the file (recommended)

The genome file (with correct sequence names) is mandatory and is called *via* –i *Full/path/to/genome*

At minima, the command line is :

python automitaur.py –i Full/path/to/genome

0-2 Parameters and options

Mandatory parameter: the genome file in fasta format

-i or –input *Full/path/to/genome*

Default parameters

-p or --prog: Program for IR search: **IRF** (if not working, will automatically shift to blast)

-p IRF or –p BLAST

Specify program if not default when re-running part of the pipeline (see below)

--TIRs: Size of TIRs: between **11** and **100** bp

--TIR min max

--MITE: Size of MITE: between **80** (whole MTE) and **750** (internal sequence)

--MITE min max

--minseq: minimum number of copies: **2**

--minseq 100

-m or --mismatch: number of mismatch in TIRs: **2** (for BLAST)

-m 4

-w or –word: size of word for BLAST search: **4**

-s or **–splitsize:** size of genome fragmentation for BLAST analysis: **100 000** (+ 5% overlap)

--cpu : number of cpu to use: **maximum**

0-3 Skipping steps

--TRF: run the analysis without prior masking of the genome

--launch: run the IR search and all subsequent step (skip the masking if already done. For running the analysis on non-masked sequence, use –TRF) [STEP2.1]

--irfblast: filter the IRF raw results with new parameters [STEP2.2] (TO DO - parameters)

--filter filter the blast-like format file of result with specified parameters [STEP2.2]

--extract: re-extract sequences (with and without flanking sequences). All subsequent steps are done [STEP3.1]

--cluster: re-cluster extracted sequences, with the specified parameters. All subsequent steps are done [STEP3.2]

--cons: Make consensus of first round of clusters, and clusterize consensus to group (too) closely related clusters [STEP 7]

--autonomous: search for autonomous (longer) copies (needs BilanTIR file) [STEP 9]

--concat: not yet implemented

--rf : not yet implemented

1- Genome masking

The genome masking is performed using Tandem Repeat Finder. This step is optional but will be performed by default.

Masking the genome

By default, the genome will be masked, the masked genome (outfile named *genome_fileM*) will be created in the same directory as the initial genome file. A subdirectory named *genome_fileM.dir* will be created and will contain all subsequent files. File generated by trf are stored in a subdirectory called 'TRF". The command-line run is the following:

```
trf full/path/to/genome_file 2 7 7 80 10 50 500
```

Refer to the help for trf online for the different parameters. If you want to change them, you will have to dive into the code (in script Mask.py file - see the trfparam variable (line 9), which contain all the mandatory arguments passed through trf excluding input file.)

Skip the genome masking step

You can skip this step with the argument --TRF in the command line. In this case, a subdirectory named *genome_file.dir* will be created and will contain all subsequent files. (if genome is masked, the subdirectory *genome_fileM.dir* is created instead. Hence you can try both and store results for both.). If those directories already exist, no error is raised but be aware that all old files with the same name as the new ones will be erased without any warning.

2- Searching for inverted repeats (TIRs)

2-a : The different IR search strategies.

You can choose to search for TIRs using Inverted Repeat Finder (irf) or using a Blast strategy. You must indicate your choice in the command line by: -p IRF, or –p BLAST. By default, IRF will be chosen. A subdirectory with the program name is created and will contain all subsequent files. You can then have the results from the two strategies at the same time on your computer.

BLAST

For the Blast strategy, a blastn search is made with the input sequences as both query and subject. For rapidity, each contig in the input file is split in 105 000 nt-long sequences and save independently in a temporary file. Blast are successively launched on each sequence_file and the result is saved in the 'BLAST' directory as 'sequence_file.BLASTout' files. All the cores of the computer will be used by default.

You can change the size of the split sequences by adding option –s or –splitsize followed by the size of the subsequences.

(ex: python automitaur.py –i input_file –p BLAST –s 10000). In this example, all the sequence will be cut in 10 500 bp-long subsequences. (an overlap of 5% is automatically generated to be sure not to miss copies).

Please note that if you're searching for elements longer than 500 bp, it may be better not to split the genome in less than 10 000 bp long subsequences.

Each subsequence is then BLASTed against itself. The word size used by BLAST can be set using –w. A word size of 4 is set by default. Filtering for low complexity is disabled, the e-value threshold is set to 50, and the task 'blastn-short' is used if using BLAST+

This step is parallelized and will use the maximum cpu available. You can restrict the number of cpu to use with the option --cpu [integer].

The resulting files are stored in the BLAST directory as "subsequences-name.BLASTout". When Blast are done, all the files are concatenated into a "BLAST.out" file, keeping only non-redundant hits, and eliminating self-hits.

Genome splitting and concatenation are performed by functions splitgenome() and concat(genometosplit, program, dirsuffix) in script SplitGenome.py.

IRF

Inverted tandem repeat is a powerful program to search for large inversions of sequence. It seems to be also efficient for small inverted repeat (above 18). (But apparently won't run on 64-bits Linux computers)

The search is launched with the command line:

irf full/path/to/genome + 2 3 5 80 10 40 X X -d -f -h' .

Refer to irf web page for parameters. If you want to change them, you will have to dive into the code (in script automitaur.py file - see the optionsIRF variable (line 117)

The IRF output is then formatted as a BLAST file (called IRF.out, in a IRF directory) (function IRFBlastize in script SplitGenome.py). At this step, a first filter is applied to discard TIR pairs that contain more than 90% AT or more than 90% GC.

Finding inverted repeats on an already masked genome.

If the genome was previously masked using automitaur and you just want to re-run a program or the other one, you can use the option --launch. --launch just skips the masking step because it considered it has already been done (presence of TRF directory). It will search for a genomeM.dir directory.

If the genome was masked by other means, you just run automitaur as if you did not want it to be masked (option --TRF) giving the masked genome as input genome (see above).

2b : Filtering according to options setting.

Hits in BLAST files are then filtered to keep only those that conform to the options of TIR size, MITE size and number of mismatches allowed between TIRs. By default, TIR sizes are between 11 and 100, MITE sizes between 80 (all element) and 750 (internal sequence), with 2 mismatches allowed (for BLAST only). Those options can be changed by –TIRs [integer integer], --MITE [interger integer] and –m (or --mismatch) [integer] options.

Example:

python automitaur.py –i Full/path/to/genome –p BLAST --TIRs 20 150 --MITE 100 500 –m 3

Filtering a BLAST.out or IRF.out file with different options

If the IR search is already performed and you want to test other parameters (TIR size, MITE size of mismatches number), you can run directly this step (along with subsequent ones) with the option --filter.

Be aware that all results previously found with different options will be erased, unless you backup the BLAST or IRF (PROGRAM) directory elsewhere.

In each PROGRAM directory, an option file is stored with the options used. At least you should backup all the directories within the PROGRAM directory.

3 : Extracting and clustering sequences.

3.1 Extracting sequences

Sequences bordered by two TIRs are extracted from the genome file. For each hit, the sequence, the sequences plus 60 nt flanking sequences on each side, and TIR sequences (useless!) are extracted and saved as fasta files. (PROG.allf.pba.seq, PROG.allf.pba.seqfl and PROG.allf.pbaITR.seq, respectively)

Sequence names are in the form: scaffold_StartPosition_EndPosition.

At this step, a filtering is performed to remove sequences with TIRs that contain more than 80 % or less than 10 % of any of the 4 nt, or N.

Re-extracting sequences.

It is usually useless, unless you manually removed some files and want to start again without doing the IR search and the filtering again (restore the initial sequence files). This can be done with the option --extract. All the subsequent steps will be run again.

Example:

python automitaur.py –i Full/path/to/genome –p BLAST --extract --TRF

3.2 Clustering sequences

Clustering sequences are done using USEARCH. For this, sequences without flanking are first sorted by decreasing length, and clustered with an identity threshold of 0.7 in at least 50% of the target and query sequences.

The command lines are:

```
uclust --quiet --sort sequence_file  --output  sequence_file.sorted
```

```
uclust  --quiet  --cluster  sequence_file.sorted  --uc  genome_file.pba.seq.uc  --id  0.70  --targetfract 0.5 --queryfract 0.5 --rev
```

The result file (.uc) is then filtered to keep only clusters with at least the minimum number of sequence required. This is performed by the script ucfAur.py. By default the minimum of sequences is set to 2 but can be increased with the option --minseq [integer].

A directory "Clusters" is created, and for each cluster with at least the minimum number of sequence required, a fasta file containing the sequences + flanking sequences is stored (ClustFxxxF.seq).

If the "Clusters" directory already exists, it will be erased.

A directory ClustersExclu0 is created (if it does not exist) for all clusters with less than the minimum sequences required. Furthermore, all excluded sequences are stored in a uniq fasta file (Exclu.seq)

For each cluster files in "Clusters", sequences are aligned with mafft and the alignment is saved as .afa file (script multimafft.py). This file will be modified afterwards (aligned)

Re-clustering if you want to change – minseq.

You can use the – cluster option. All subsequent steps will be performed and the Clusters directory will be erased. Backup it elsewhere before proceeding if you want to keep a trace.

5 : Removing duplicates

60 bp of flanking plus 60 bp terminal sequences at each end are extracted for each sequence in clusters, and clustered at an identity threshold of 0.90. Clusters are suppose to represent duplicated sequences. Only one representative sequence is kept as the independent type.

The sequence is kept if both ends are kept by the both clustering processes at 5' and 3'. This result in two new files in the program directory : Prog.allf.pba.seq.uniq contains all independent copies. Prog.allf.pba.seq.dup contains all discarded redundant copies

Then for the different cluster, a new fasta file containing only aligned independant copies (with flanking) is created in Clusters directory (ClustFxxxF.ifl.afa). In this file, all position made of gap only are deleted. If there is less than the minimum required number of sequences among independent sequences, all associated files in “Clusters” are moved in ClustersExclu1

All this is performed by the function rmdup() in the script remove_duplicate.py

At this step, a summary file is created (‘BilanNbSeq') that stores for each cluster the total and independant copy number, in the PROG directory

6 : Finding precise boundaries

As TIR positions may be different between copies in a cluster, because of mismatches for example, the TIRs are determined precisely at this step. TIR positions are stored in BilanTIR and TIR sequences are stored as fasta in BilanTIR.fa

For this, each alignment is aligned with the reverse-complement alignment (function createfiles in script FindBoundaries.py), and aligned sequences (corresponding to TIRs) are determined (function AnalyseCluster). For more details, dive into the code of email-me!

For each cluster, a .fl53 file is created containing 30 bp 5’ flanking sequence, 3 5’ TIR bp, 3 3’ TIR bp and 30 3’ flanking sequence.

For each cluster, a ClustFxxxF.afa file is created which contains the clean independent sequence (without flanking)

If no good TIRs are found in the cluster, all associated files in “Clusters” are moved in ClustersExclu2. A ListTIR is created in the Clusters directory and contained start position of 5’ and 3’ TIRs relative to the alignment file, and length of 5’ and 3’ TIRs. (It seems that the initial alignment file .ifl.afa ? does not exist anymore.)

7 : Reject clusters

If too short or too many gap, all associated files in “Clusters” are moved in ClustersExclu3

8 : Make Consensus sequences

For each sequence a consensus sequence is derived from the independent clean fasta file (.afa), and stored in the file Clust.Cons.fas in Cluters directory (This is performed by the script ConsAur.py). The TIR sequences from each consensus is stored in *PROG/BilanTIR.fa* as well as some info in *PROG/BilanTIR*.

Consensus sequences are then clustered again at an id threshold of 0.70 that allow to gather clusters that correspond to more divergent copies from the same subfamily, that were not kept into the same cluster at the first clustering step.

If two consensus are clustered. all files corresponding to the first are updated with files from the second. Sequences files are not aligned again, so the alignment may be impaired. The new cluster keep the name of the first one and all files associated to the kept clusters are stored in the Cons directory.

9 : Find TSD

The .fl53 file created for each cluster at step 6 is used to find the most probable TSD for each cluster. For this, direct repeat and TA are searched at the boundaries of each sequence. If direct repeats of several lengths are found, the most frequent size is kept (or the longer). If a TA is more often than direct repeat, it is kept as the potential TSD. (note that TSD longer than TA but containing TA (such as piggyBac TTAA) are considered as TA TSD)

Result are stored in the BilanTSD file (for each cluster are reported the length of most frequent length of DR, the number of occurrence of this DR, the number of sequence, the finding of a TA, the number of occurrence of TA and the number of sequence.

This step is performed by the function analyseTSD in the script FindTSD.py.

10: Find autonomous element

This step is realized by the script autonomous.py (function autonomous). For each cluster, consensuses TIR (all stored in the fasta file BilanTIR.fa, are searched by BLAST in the genome. Hits are filtered to keep only contiguous pairs that are 1500 to 5000 bp apart (BlastTIRAnalyse.blastTIR function). Sequences are stored in the bpia.seq file in Auto directory. (solo TIRs and soloTIRs plus flanking are stored as coordinate in case of – bsia and bsiaf)

Then all sequences in bpia.seq are sorted by TIR (cluster). A new clustering is also done with all sequences (USEARCH-clustering)

Results (number of clusters for which potential autonomous elements have been detected) are given in the terminal for each method.

In the Auto directory, clusters of longer element are stored according to the TIR classification.

11: Summary files

List of MITE: BilanCopies.txt

Summary file: FinalBilan

Consensus sequences fasta file: Cons/Clust.Cons.fas

Cons directory: where final cluster sequences are located

Cluster directory: where all kept sequence are located, but may contains different clusters corresponding to the same family of MITEs (prior to re-clustering of consensuses – step 8)

For each cluster (in Cluster or Cons directory):

4 files are generated:

.afa: (more or less) aligned independent sequences without flanking regions

.allafa: all sequences including duplicated ones

.ifl.afa: independent sequences with 60 bp flanking sequences each side

.fl53 : 30 bp flanking sequence from each side (emcompassing 3 bp of each end of the element). Useful for empty site analysis.

Excluded Clusters:

Excluded Clusters are stored in separate directory (Exclu0:Exclus4) according the the exclusion cause:

Exclu0: Not enough sequences in cluster

Exclu1: Not enough sequences in cluster after duplicates removal

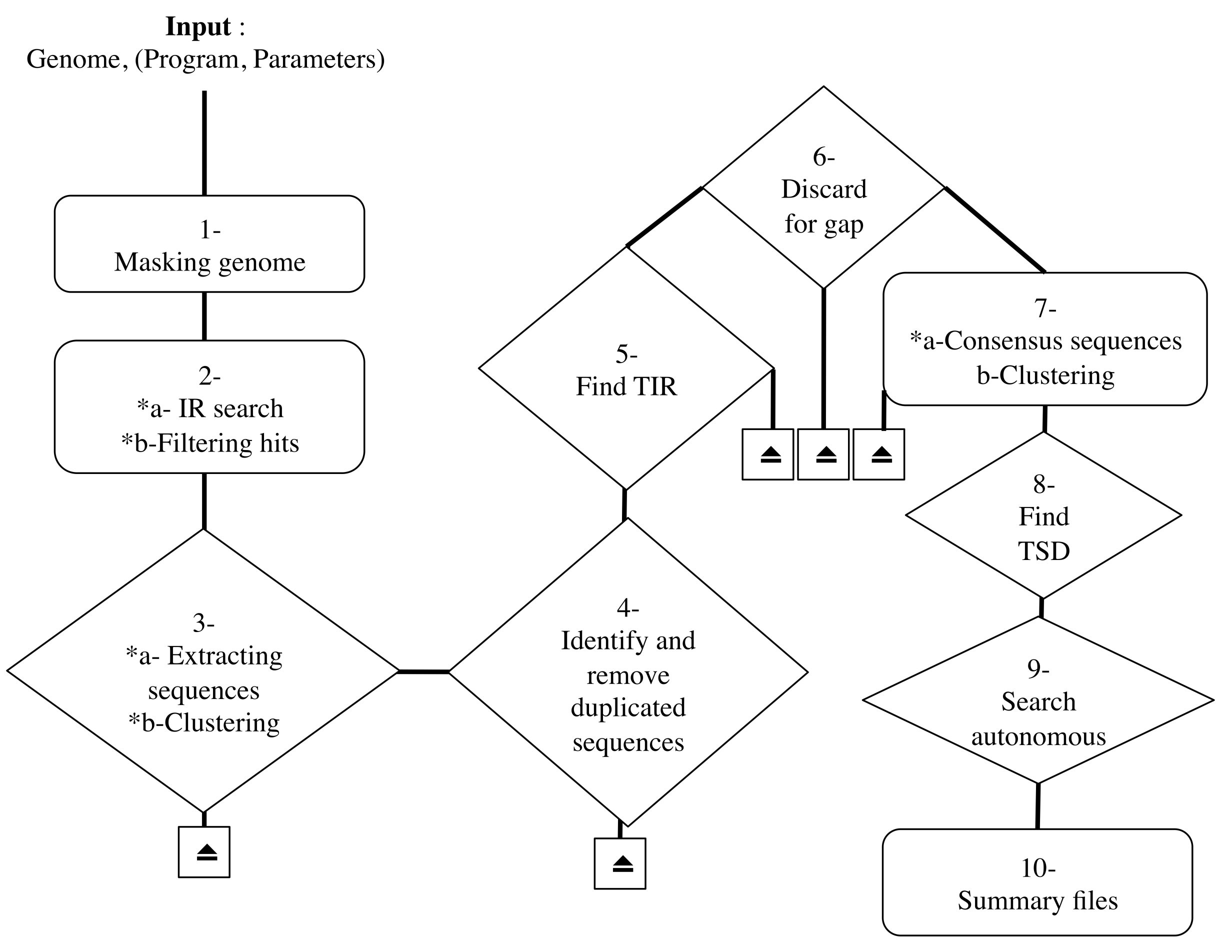
Exclu2: Good TIRs not found

Exclu3: Too many gaps between sequences

Cons.Exclu0: ?

All these files can be removed after running CleanAutomitaur.py

III- Flow chart


Input :
Genome, (Program, Parameters)
1- Masking genome
2- *a- IR search *b-Filtering hits
3- *a- Extracting sequences *b-Clustering
4- Identify and remove duplicated sequences
5- Find TIR
6- Discard for gap
7- *a-Consensus sequences b-Clustering
8- Find TSD
9- Search autonomous
10- Summary files