Biogo Command Line Applications User Manual

Biogo written by: Dan Kortschak
Document organised by: Lu Zeng

Biogo (https://github.com/biogo/examples/) is a bioinformatics library for the Go language. Biogo/krishna and Biogo/igor are de-novo repeat family identifying and annotation packages, which employ computational methods for identifying repeat element boundaries and family relationships from sequence data. Biogo assists the runs of krishna and igor given a genomic database and uses the output to build, refine and classify consensus models of putative interspersed repeats.

Prerequisites

1. Download Go

Available at (https://golang.org/dl). Developed and tested with version 1.5.2.

Install on Linux:
Unpack the Go language in your home directory or in a location where it may be shared with other users of your system (ie. /usr/local/).

* cp go#.#.linux-amd64.tar.gz /usr/local/
* tar -C /usr/local -xzf go#.#.linux-amd64.tar.gz
* export PATH=$PATH:/usr/local/go/bin

If you installed Go to your home directory you should add the following commands to $HOME/.profile. For example, put your go path into a directory called "work":

* export GOROOT=$HOME/go
* export GOPATH=$HOME/work
* export PATH=$PATH:$GOROOT/bin

For more installation details, follow the instructions on the Go installation page.

2. Git

To perform the next step you may have Git installed. (Check that you have a git command before proceeding.)

If you do not have a working Git installation, follow the instructions on the Git download page.

3. Download Biogo Packages

Download krishna and igor packages from github.

* go get github.com/biogo/examples/krishna
* go get github.com/biogo/examples/igor
* go get github.com/biogo/examples/external

4. Install Biogo Packages

Install packages from Biogo in you home directory, for example directory called "work".

* cd work/src/github.com/biogo/examples/krishna
* go build -o $HOME/work/bin/krishna-matrix matrix.go
* go build -o $HOME/work/bin/krishna krishna.go
* cd work/src/github.com/biogo/examples/igor
* go build -o $HOME/work/bin/igor igor.go
* go build -o $HOME/work/bin/seqer seqer.go

5. Install CENSOR

Install censor to screens target genomes against a reference collection of repeats libraries with masking symbols, as well as generating a report classifying all found repeats. CENSOR needs wu-blast and Bioperl.

The version is now available at CENSOR download page.

Example run

In this example, human genome were downloaded by chromosomes (24 chromosomes) from UCSC into files called chr-*.fa.

1. Use krishna-matrix to run alignment between human sequences as a matrix, it will run self alignment first, and then run pairwise alignment.

The default minimum hit length for krishna (-dpid) is 400bp, and minimum hit identity (-dplen) is 94%. The smaller number of parameters you use, the longer time and bigger memory will be needed.

If you want change these parameters, just add "-dplen=*bp" and "-dpid=0.*" in 'matrix.go' under krishna source code: lines 102 and 132 (codes begin with "cmd := exec.Command").

Change the work directory that stores temporary files generated from krishna to you own directory (line 64 in matrix.go), Then rebuild 'matrix.go'.

Notes: You need to rebuild the code every time when you did any changes.

Now runs the job:

* cd /data/rc003/lu/human (directory contains the sequences)
* krishna-matrix chr-*.fa

If the genome sequence were very big and consisted by multiple contigs or scaffolds (>200MB), split it into smaller files (using bundle.go).

* go build -o $HOME/work/bin/bundle bundle.go
* bundle -bundle 50000000 -in seq.fa

-bundle: Specifies the sum of sequence length in a bundle. (default 20000000, 20MB).
-in: the genomes sequences need to split.
Then run krishna job.

2. Use igor to take pairwise alignment data produced by krishna, and reports repeat feature family groupings in JSON format.

* cat *.gff > hg_krishna.gff
* igor -in hg_krishna.gff -out hg94_krishna.json

3. seqer returns multiple fasta sequences corresponding to feature intervals described in the JSON output from igor.

seqer will also produce fastq consensus sequence output from one of MUSCLE or MAFFT. gffer converts the JSON output of igor to gff.

* gffer < hg94_krishna.json > hg94_krishna.igor.json
* cat chr-*.fa > hg19v37.mfa
* seqer -aligner=muscle -dir=consensus -fasta=true -maxFam=100 -subsample=true -minLen=0.95 -threads=12 -ref=hg19v37.mfa hg94_krishna.igor.json

-fasta: Output consensus as fasta with quality case filtering
-maxFam: maxFam indicates maximum family size considered (0 == no limit).
-minLen: Minimum proportion of longest family member.
-threads: Number of concurrent aligner instances to run.

Benchmarks

Genome	Krishna Threads	Genome DB Size	Krishna run time (hh:mm)	Igor run time (hh:mm)	Seqer run time (hh:mm)
Human	4~8	3.0G	<350	128:30	2:23
Bearded Dragon	4~8	1.8G	<48	92:40	<4
Chicken	4~8	1017M	<24	<4	<1
Opossum	4~8	3.5G	<288	61:48	4:52
Platypus	4~8	2.0G	<340	191:34	10:16
Echidna	4~8	1.9G	< 360	12:43	9:02

* Analysis run on a 512GB RAM, machine running Red Hat Linux

Repeat Library Annotation

The previous steps generate the repeat consensus sequences from the human genome, now we are going to annotated these consensus with repeat class.

Annotate consensus sequences

Notes: All Jave scripts used here need to specify the directories where your data is and where you want your output in

1. Annotate consensus sequences with repeat families.

Use censor to annotated consensus sequences with Repbase library.

* cd consensus
* cat *.fq > ConsensusSequences.fa
* censor -bprm cpus=8 -lib ~/Verterbrates.fa -lib ~/our_known_reps_20130520.fasta ConsensusSequences.fa

The censor outputs usually have 5 files: ConsensusSequences.fa.map, ConsensusSequences.fa.aln, ConsensusSequences.fa.found, ConsensusSequences.fa.idx, ConsensusSequences.fa.masked.

2. Classify consensus sequences.

ConsensusSequences.fa and ConsensusSequences.fa.map were required in this step. You will also need to specify the directories where your data is and where we want your output in the java code, then compile and run it.

* javac ClassifyConsensusSequences.java
* java ClassifyConsensusSequences

You should get 5 output files: known.txt, partial.txt, check.txt, notKnown.fa, notKnown.fa.gff.

3. Filter potential protein sequences.

In this step, you will need to run the perl script reportsJ.pl (perl reportsJ.pl). This uses wu-blast blastx, you will also need uniprot database. reportsJ.pl will ask for a list of databases to process, "libs.txt". We need file notknown.fa and notknown.fa.gff, put libs.txt into current directory.
Notes: reportsJ.pl contains three parts: used to identify uniprot, GB_TE, or retroviruses. You can comment other two parts, and run one of each parts separately in the same time, to save the running time.

4. Get proteins information from consensus sequences.

Another java program GetProteins.java will be used. It need two input files: notKnown.fa, notKnown.fa.spwb.gff (Generated from previous step).

* javac GetProteins.java
* java GetProteins

You will get 2 output files: proteins.txt (a list of families that have been identified as proteins and the proteins they have matches); notKnownNotProtein.fa (a fasta file of the families that have not yet been classified).

5. Check simple sequential repeats (SSR).

Then check any existence of SSR in the unknown sequences, using phobos.

* phobos -r 7 –outputFormat 0 –printRepeatSeqMode 0 notKnownNotProtein.fa > notKnownNotProtein.phobos

6. Identify the sequences that are SSRs from the phobos output.

The phobos output will be used to identify SSRs: notKnownNotProtein.phobos.

* javac IdentifySSRs.java
* java IdentifySSRs

You will get a file called: SSR.txt

7. Generate annotated repeat library.

10 input files are require to generate a repeat library:
1. ConsensusSequences.fa
2. ConsensusSequences.fa.map
3. notKnown.fa.tewb.gff
4. notKnown.fa.ervwb.gff
5. protein.txt
6. known.txt
7. GB_TE.21032016.fa
8. all_retrovirus.fasta
9. SSR.txt (if you do not have this, leave the definition in, it will cause error messages, but will not stop the program nor effect the results.)
10. LA4v2-satellite.fa (you do not have this, or equivalent, as you didn't have nay satellites, but leave the definition in-it will cause error messages, but will not stop the program nor effect the results.)

* javac GenerateAnnotatedLibrary.java
* java GenerateAnnotatedLibrary

Then you will get a library called "*_Library.fasta", you can also change this to whatever you want to call your library

Benchmarks2

Genome	Consensus sequences size	Censor first run time (hh:mm)	reportJ.pl (hh:mm)	phobos run time (hh:mm)
Human	38M	5:14	19:30	<00:05
Bearded Dragon	88M	2:41	< 192	<00:05
Chicken	18M	3:01	<24	<00:05
Opossum	60M	13:17	<48	<01:00
Platypus	162M	34:34	<192	<01:00
Echidna	59M	36:40	105:12	01:54

* Analysis run on a slurm machine with 4~~16 cpus, and 8GB RAM, machine running Red Hat Linux.

Finished! Good Luck!!!