WGAC

Whole genome assembly comparison of duplication originally described in Bailey et al. 2001.

Inputs

- species name
- path to FASTA sequence(s) to be processed
    - either a directory of chromosomal FASTA files or a single genomic FASTA

Setup

Create a new working directory.

```
mkdir wgac
cd wgac
```

The genomic sequences should already exist somewhere else in a standard library. If they don't, download them from UCSC or somewhere similar and create a library for them.
For sequences that do already exist, create a local symbolic link called "fastawhole" to those sequences. Calculate sequence lengths for later analysis.

```
ln -s /net/eichler/vol2/eee_shared/assemblies/canFam2/chromosomes fastawhole
fasta_length -d fastawhole/ -o fastalength.log -f
```

Mask sequence(s)

If RepeatMasker output is already available for your sequences (as it commonly is for species assemblies from UCSC), it is simpler to mask your sequences before splitting them into 400 Kbp segments. Then, after splitting, you can recreate .out files for each segment using the lowercased letters.

Create "fuguized" sequences

Break (fractionate) the FASTA records into smaller pieces. The "-s" flag specifies size in base pairs for each piece.

```
mkdir fasta
fasta_fractionate -f fastawhole -s 400000 -o fasta
```

If your sequences are already masked, you just need to create .out files corresponding to each 400 Kbp segment as follows.

```
mkdir mask_out
ls fasta | xargs -i perl ~jlhudd/wgac/maskOutGenFromLowCase.pl fasta/{} mask_out/
{}.out
```

If your original sequences are not masked, mask repeats in the fractionated sequences using the quickest RepeatMasker setting. It takes about an hour to mask ~10,000 segments of 400Kb each. The masking script creates output in the current working directory under the "Masking" subdirectory.

```
qsub -q all.q ~jlhudd/pipelines/repeatmask/repeatmask.sh \
  -s <species> \
  -i `pwd`/fasta \
  -q rush
```

Create fugu-like sequences (without repeats) using FASTA and RepeatMasker .out files. RepeatMasker files should be 1-based. For the human genome this step takes 13 minutes and fugu directory should be about 1.6 Gb.

```
mkdir fugu
fasta_fuguize_batch.pl -f fasta -r Masking/step1/mask_out -o fugu
```

BLAST fuguized sequences

Make a BLAST database from a single huge FASTA file.

```
mkdir blastdb
dir_cat fugu blastdb/bofugu
formatdb -i blastdb/bofugu -o F -p F -a F
rm -f blastdb/bofugu
```

Run BLAST on the fuguized sequences. Run this step concurrently with the following self-blast step.
The SGE BLAST script takes the following arguments:

- -b BlastDB source path (default: blastdb/bofugu)
- -t BlastDB temporary directory (default: /var/tmp/blastdb)
- -i input directory (default: fugu)
- -o output directory (default: blastout)

If you are in the WGAC working directory, the only argument you should need to submit is the temporary directory path. This path should be based on your username so you can easily cleanup after yourself later.

```
mkdir blastout
qsub -q all.q ~jlhudd/wgac/blast64.sh -t /var/tmp/jlhudd/blastdb
```

When BLAST is complete, parse the output.
First check for files of nohit based on the size and remove these files. File size will be less than 1300 (containing only a header and message no hits found).
This command will produce nonsense output if $size is empty and therefore still less than 1300.

```
perl /net/eichler/vol5/home/tinlouie/wgacscripts/rmNohitfiles.pl blastout
```

Check for files with problem aligns (e.g., files without blast output message Matrix: blastn matrix:30 -80). Sometimes the blast output file is truncated at the end. In this case you have to either fix the file by adding summary to the end of the file or rerun the blast for the specific file (this latter step is probably easier).

```
find blastout -type f -exec grep -L 'Matrix' {} \;
```

Parse files with a seed size of 250 bp and identity 88%. You can change the parameters, seed size and identity. Sometimes if the repeats are not removed well, such as with elephant and zebra finch, you may want to use a seed of 500. This step disregards self hits and outputs data/bo_self.parse.

```
mkdir -p data

blastparser42.pl -in ./blastout \
  -out data/bo_self.parse \
  -noprevq -v \
  -output '-ssort=>name, -hsort=>qb' \
  -filter '-min_bpalign=>250, -min_fracbpmatch=>0.88, -max_%gap =>40, -
no_subject_self => yes, -no_doubleoverlap=>score'
```

Self-BLAST

Run self-blast (concurrent with previous BLAST). **This step can only be run on a specific architecture (Sun workstation).**
Replace the letters “RYKMSWBDHV” with “N” and switch all letters to uppercase.

```
cp -r fugu fugu2
fasta_only_ATCGN.pl -p fugu2 -o fugu2 -u
```

Run webb_self, a precursor to blastz. The perl script invokes webb_self repeatedly, with different parameters. The directory “selfblast” will contain one output file per fugu file.
During the run, the script may output a message like “Segmentation Fault - core dumped” because of the parameters to webb_self. webb_self will be re-invoked, with different parameters, on the same input file. Hopefully, one of the invocations will have a good result.
If the program appears to be stuck because the last message is something like 'DOING chrX_0416.fugu…', and the corresponding output file (e.g., chrX_0416.fugu.intf) remains at size zero and unchanged for 24 hours, press control-C, and the output file should become non-zero.
The remote host only has access to volumes 1-11 so the “fugu2” and “selfblast” directories need to be located on one of those volumes.

```
mkdir selfblast
perl fugumation.pl -i /net/eichler/vol~~~/fugu2 -o /net/eichler/vol~~~/selfblast
```

When selfblast is complete, exit the Sun workstation and clean up the copy of the fuguized files.

```
# Leave Sun computer.
exit

# Clean up.
rm -rf fugu2
```

Parse self-blast output.

```
mkdir tmp
blast_lav_break_self_overlap2.pl --in selfblast --out tmp

mkdir -p data
blast_lav_hit_by_hit.pl --in tmp --out data/lav_int2.parse -
options 'MIN_BPALIGN=>200, MIN_FRACBPMATCH=>0.88, MAX_%GAP => 40,
SKIP_SELF => 0, SKIP_OVERLAP=>1'

rm -rf tmp
```

Merge parsed BLAST results

Combine results of previous two steps into one file.

```
cd data
mv bo_self.parse both.parse
sed 1d lav_int2.parse >> both.parse
cd ..
```

Defuguize regions

Defuguize a pairwise blast table (both.parse) using the original “fuguized” sequence and the RepeatMasker output. This creates a file named “both.parse.defugu” using subdirectories “fugu” and “mask_out”.
This step may take 9 hours, depending on size of “both.parse”.

```
blast_defuguize_hit_by_hit.pl -t data/both.parse -d .
```

Trim ends

Trim sequence ends.

```
cd data
mkdir step_8_mpi
mkdir step_8_mpi/defugu
```

Split the defugu file into smaller files and store those files in a new directory called “newdir”. The first argument is the name of the defugu file. The second argument is the total number of smaller files you want to have. The third argument is the number of lines in defugu file.

```
perl ~jlhudd/wgac/split.pl both.parse.defugu 300 98300
find newdir/ -type f -exec cp {} step_8_mpi/defugu \;
rm -rf newdir
mkdir step_8_mpi/trim
cd ..
```

Submit the end trimming job. If you are in the root WGAC directory (parent of “data”), you can run the following command without any flags. You may also specify any of the following flags:

- -i <INPUT>
- -f <FASTA FILES>
- -o <OUTPUT>

Make sure to request enough CPUs or this job will take a long time to run against larger data sets. This script creates /tmp/endtrim on each machine. Make sure there is no such directory on the machines before running the code by running the following command on the head node:

```
cexec eeek: ls /tmp/endtrim
```

Submit the trim ends job.

```
qsub -q all.q ~jlhudd/wgac/trim-ends.sh
```

Collect the generated trim files into one file named ParallelOutput.trim and located in data/step_8_mpi/trim.

```
cd data
perl ~ssajjadi/wgacbin/step_8_mpi/collectTrim.pl step_8_mpi/trim/
mv step_8_mpi/trim/ParallelOutput.trim both.parse.defugu.trim
rm -rf step_8_mpi
```

Generate both.parse.defugu.trim.fixed.trim
This step fixes for overlap alignment ('overlaps' in files), reduces the number of lines (one half of one percent), and takes 2 minutes.

```
blast_align_by_align_overlap_fix3.pl -i both.parse.defugu.trim
```

Generate both.parse.defugu.trim.fixed.trim.defrac
This step takes 3 minutes for the human genome. Note that the parameter 400000 was used in an earlier step to fractionate the chromosome files in fastawhole/. Adjust your value here accordingly.
At this step, check the size of `both.parse.defugu.trim.fixed.trim.defrac`. If this file is larger than 100MB, the global alignments step may not be completable and you may crash the head node.

```
blast_defractionate3.pl -s 400000 -t both.parse.defugu.trim.fixed.trim -f ../fastawhole
```

Run global alignments

For the human genome, the first stage takes many hours.
You must run the following shell script on the head node (eeek), not one of the grid nodes (e8, e10, e12, etc.). You may run the script on e1 using “conf.txt”.

```
/net/eichler/vol5/home/tinlouie/wgacscripts/wgacAlign.sh ../fasta 64bit
```

wgacAlign creates new directories: both_tmp and align_both (one is for synchronization between CPUs, the other holds output files).
cexec is stuck if both_tmp contains subdirectories (e.g. both107394/) and no subdirectory has been touched for many hours.

```
ls -lrt both_tmp
```

To kill wgacAlign use the following steps. Wait a few minutes after the kill command for both_tmp to reach a stable state.
If you have bad luck, you may have to kill Perl processes (and the compiled C program 'align') on nodes of the cluster.

```
control-Z
ps | grep wgacAlign.sh | cut --delimiter=' ' -f 1
kill -9 <PID of wgacAlign>
```

Whether or not you had to kill the script, the next step is the same:

- count the number of files in all subdirectories of align_both
- count the number of lines in the defrac file

For the human genome, Saba got 111726 and 111738, respectively. For this genome, the final count of output files ideally should be 111,737 due to header line of defrac file.

```
ls -d align_both/*/ | xargs -i ls -1 {} | wc -l
wc -l both.parse.defugu.trim.fixed.trim.defrac
```

Run the script again without the '64bit' parameter so the script runs on just one machine. For hg19, this stage should take less than 1 hour.
Remove the temporary directory to prevent the single CPU from confusing its output with multiple-CPU output. If everything went correctly with the first alignment step, this directory should be empty already.

```
rm -rf both_tmp/both*

/net/eichler/vol5/home/tinlouie/wgacscripts/wgacAlign.sh ../fasta
rm -rf both_tmp
```

There should be an increase in number of output files. For hg19, this should be close to 111,737.

```
ls -d align_both/*/ | xargs -i ls -1 {} | wc -l
```

Compute scores

Compute scores and other statistics (e.g. Kimura's). This step takes 80 minutes for hg19.
Output is a new directory “align_both_data” and a new file “both.alignscorerdata”.

```
align_scorer_batch2.pl -a align_both \
   -d align_both_data -p both \
   -o both.alignscorerdata
```

Output of the first command minus one (due to the header) should be equal to the output of the second command. For hg19, Saba got 111499 for the second command.

```
wc -l both.alignscorerdata
grep -c 'align_both' both.alignscorerdata
```
Merge files defractionated file with score and stats data. This step takes 1 minute for hg19.
```
table_merger.pl -s'row(\d+):both(\d+)' \
   -i both.parse.defugu.trim.fixed.trim.defrac:both.alignscorerdata \
   -o defrac.align
```

BLAST welder

Run the “welder”. This step takes 1 hour for hg19. The welder produces several files all with the same prefixoo.weild10kb.
The -g parameter specifies smallGap:largeGap:overlap. To join two pairwise alignments, both query and subject should have gaps < largeGap and at least on of them should be < smallGap.
```
blast_hit_by_hit_welder2.pl -i defrac.align -g 40:10000:20 -o oo.weild10kb
```
Create two files named blah.blah.cull. This step takes less than 1 minute for hg19.oo.weld10kb.join.all.cull will be used in next step.
```
table_line_culler2.pl -p oo.weild10kb.pieces.all -t'$c[27]>=0.90 && $c[64]>=1000'
table_line_culler2.pl -p oo.weild10kb.join.all -t'$c[27]>=0.90 && $c[22]>=1000'
```
WGAC calculations should be complete now. What follows are steps related to analysis of these data.

Create summary files

Create a “super dup” file.

```
perl /net/eichler/vol5/home/tinlouie/wgacscripts/makeTrack.pl  \
   oo.weild10kb.join.all.cull > GenomicSuperDup.tab
```
Make sure there isn't redundancy in the super dup file.
```
perl ~jlhudd/ps/wgac/rmRedunInSUperDup.pl GenomicSuperDup.tab
```