2bRAD pipeline for pooled DNA samples

Author: Evan Durland

Updated: October 17, 2018

Project: Oyster larvae pool-seq 2bRAD

Brief: In this example, we were analyzing a 2bRAD library from pooled DNA samples of larvae of *Crassostrea gigas*. **note 2bRAD markers are ALL uniformly 36bp in length, for other (longer and size-variable) marker types, adjustments to the script will be necessary for quality reads.*

Bioinformatic packages follow Eli Meyer's pipeline. Scripts from his pipeline have been copied to a local directory and linked within (name changes are unique to this pipeline). Make sure all scripts are in your $PATH for execution.

Computing infrastructure: CGRB @ OSU

Step 1: load fastq files to cluster.

Step 2: Trim, quality filter and remove primers using custom script. First, list files for processing:

```
$echo *.fastq > files.txt
```

Now put one per row:

```
$sed -i -e 's/\s/\n/g' files.txt
```

Next, run a truncation script; TruncateFastq.pl and two filtering scripts: QualFilterFastq.pl and an adaptor filter; bbduk.sh. After tinkering with settings, I settled on a quality filter with base quality phred **>20** and max poor quality bases per read of **10**. For adaptor filtering, you need a file with illumina adaptor sequences, in FASTA format. First we'll need to make a 'stats' folder to put some output into:

```
$mkdir stats
```

These three steps can be run together with the custom bash script TruncQualAdapt.sh with two input parameters: the list of files (`files.txt` from above) and location of the adaptors.fasta reference. The adaptor filter step is memory hungry so give it plenty of memory (~20 G) to play with. These processes (and future ones) are run as a batch command on the compute node so as to not overwhelm the login node. On Oregon State University's CGRB infrastructure, they are sent as a Sun Grid Engine (SGE) batch command as follows. Your computing infrastructure may require adaptations to how jobs are submitted.

```
$SGE_Batch –c "TruncQualAdapt.sh files.txt adaptors.fasta" –r JOB_NAME –q
COMPUTE_NODE –m 20G
```

This step is likely to take some time so run it overnight if possible. After it is done, extract read quality info from output files (grep is useful here). Next, index the reference with ExtractSites.pl. Inputs are the genome (-i), the enzyme pattern to search for (-e) and the output file name (-o):

```
$ExtractSites.pl –i oyster_genome.fna –e BcgI –o oyster_genome_BcgI.fasta
```

This generates a smaller genome with only the BcgI sites so that mapping is more efficient. Now we can map the .fasta files to the indexed genome. Using gmapper from the SHRiMP package we can run the shrimp.sh program on a list of files to be mapped:

```
$SGE_Batch –c “shrimp.sh files.txt oyster_genome_BcgI.fasta” –r JOB_NAME –q
COMPUTE_NODE
```

The main parameters for this mapping are:

```
gmapper --qv-offset 33 -Q --strata -o 3 -N 1 FILE.fasta REFERENCE.fasta >
OUTPUT.sam
```

Where qv-offset is the ASCII offset of quality values, -Q designates fastq files, -o is the maximum allowed # hits per read (avoid duplicated genes/multiple alignments) and -N is the number of threads (cores) to run on. This also takes some time so it is best done overnight. After mapping is completed, the system log will contain the mapping details, it is best accessed with grep:

```
$cat map/map.o##### | grep –A 8 General
```

Which will generate something like this:

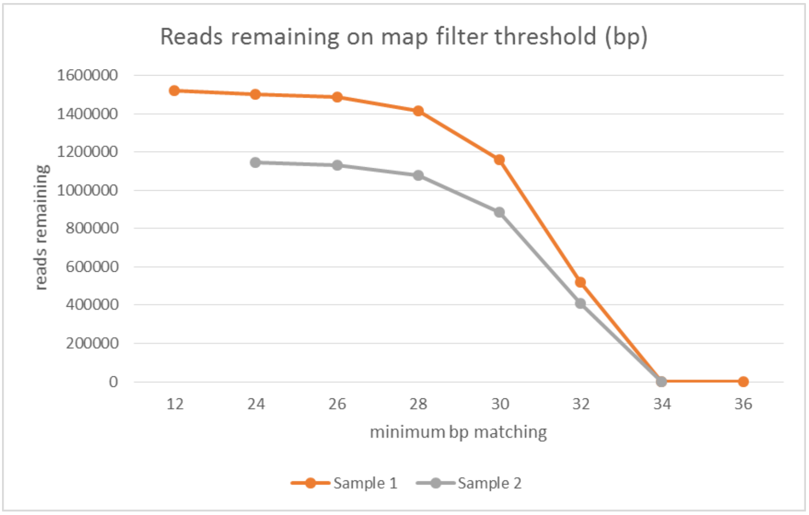
```
General:
        Reads Matched:          2,989,686    (55.4575%)
        ... with QV >= 10:      1,203,517    (22.3248%)
        Reads Dropped:          0    (0.0000%)
        Total Matches:          6,344,654
        Avg Hits/Matched Read:  2.12
        Duplicate Hits Pruned:  0

    Memory usage:
```

For this example we matched ~2.9 million reads (55%), of which 1.2M were high quality (QV>10). No reads were dropped for quality because we did quality filtering previously. **Avg Hits/Matched Read** is the average number of alignments for each read (we capped it at 3) and **Total Matches=Reads Matched x Avg Hits/Matched Read** (e.g. 6,344,654 = 2,989,686 x 2.12). Next we’ll filter these mapped reads for low alignment quality:

```
SAMFilter_3.pl –i INPUT.sam –m 30 –o OUT.filt.sam
```

Where –m 30 means you want a minimum of 30bp matching the read location. I played around with settings from 12 to 36:



You can see that requiring > 28 matches per read is desirable to weed out some of the poorly mapped hits but when we get up to 34 and above, there are nearly no matches. I ran an entire set of files on 32 bp matches (recommended as default) and was left with a lot of reads but no polymorphisms. Consequentially, I decided to run with 30bp matches for this data. This may vary by run and per species being studied. I'd recommend exploring this to customize your scenario.

Now that you have filtered SAM files, we need to convert that to the number of base calls per loci:

```
SAMBasecaller_3.pl -I INPUT.filt.sam -r oyster_genome_BcgI.fasta -c 5 -o
OUT.bc.tab
```

This will compare each read to the reference genome, filter loci with < 5 (-C) reads and generate a tab delimited file like this:

```
Tag                    Locus  Ref   A    C    G    T
NC_001276.1_473_508_F    2     G    0    0    6    0
NC_001276.1_473_508_F    3     A    6    0    0    0
NC_001276.1_473_508_F    4     A    6    0    0    0
NC_001276.1_473_508_F    5     T    0    0    0    6
NC_001276.1_473_508_F    6     C    0    6    0    0
NC_001276.1_473_508_F    7     A    6    0    0    0
NC_001276.1_473_508_F    8     A    6    0    0    0
NC_001276.1_473_508_F    9     T    0    0    0    6
NC_001276.1_473_508_F   10     A    6    0    0    0
```

Following this, we just need to concatenate all the files with CombineBaseCounts.pl:

```
CombineBaseCounts.pl *files*.bc.tab > combined.tab
```

This file looks like this:

```
       Tag                     Locus  Ref   MA1-D22B.   MA1-D2.   MA2-D2.
```

```
NC_001276.1_473_508_F     1      A      0/0/0/0     16/0/0/0   0/0/0/0
NC_001276.1_473_508_F     2      G      0/0/6/0     0/0/57/0   0/0/0/0
```

Lastly, the custom Python script `Combinator.py` will take the above file and simplify it to major/minor alleles, filter on coverage thresholds (50-1000 default) and remove loci where minor reads are 2 or less:

```
Combinator.py INPUT.tab OUTPUT.txt
```

The result is something like this:

```
Tag                         Locus  Ref  alleles  MA1-D22B.  MA1-D2.  MA2-D2.
NC_001276.1_7728_7763_F       3     C     C/A       69/5     338/0      NA
NC_001276.1_7728_7763_F      12     G     G/A       74/0     308/30     NA
```

The entire process can be streamlined with `sam_to_counts.sh`

`From here you should be ready to export to R and begin to analyze!`