GCAT-SEEK Eukaryotic Genomics Workshop

Tues

PM Module 1. Genome assembly I: Data quality and genome size estimation using Jellyfish

Module 2. Genome assembly II: Data quality and error correction using Quake

Module 3. Genome assembly III: Assembly algorithms

Weds

AM Module 3. (cont)

Module 4. Genome annotation with Maker I: Introduction and repeat finding

PM

Module 5. Genome annotation with Maker II: Genome annotation with Maker II. Whole genome analysis

Module 6. Whole genome annotation: Misc.

Thurs

AM Module 7. Genome Genomics using CoGe

Module 8. Stacks

PM Module 8. Stacks (cont)

Module 1. Genome assembly I: Data quality and genome size estimation using Jellyfish

Background

An underlying assumption of whole genome sequencing projects is that reads are randomly distributed around the genome at approximately equal coverage. If a genome is sequenced at 100x coverage, one expects each nucleotide to be covered 100 times by randomly distributed reads. A common strategy in sequence analysis is decomposition of reads into shorter kmers. While not all 100 bp reads in the genome will be generated by a sequencer even at 100x coverage, each 100bp (L= read length) read will generate (L-k+1) kmers. For example, the read GCAT, would decompose into (4-3+1) or two 3mers, GCA and CAT.

*Q. How many 3mers would the read GCATSEEK produce?*

Decomposition of reads into small kmers allows data quality to be inferred from histograms of kmer counts. For a 1Mbp genome, to generate 100x coverage of 100bp reads (L=100), one would generate 100M reads. These reads could be decomposed into 15-mers (K=15) which would have an average genome coverage of [L-k+1)/L\*expected coverage per nucleotide] = (100-15+1)/100\*100 = 0.86\*100 = 86X coverage (Kelly et al. 2010). The average kmer coverage is lower than nucleotide coverage because not all reads covering a nucleotide will include all kmers for that nucleotide.

Kmers follow an approximately normal distribution around the mean coverage. The first peak between one and the dip between the peaks represents mainly error kmers. While low coverage areas will have low kmer counts, most areas of the genome would have a much higher coverage. The first peak is primarily a result of rare and random sequencing errors in the read. Below we show how to get estimates of genome size, and error rate from these graphs.

The program Jellyfish (Marcais and Kingsford 2011) is a fast kmer counter. Below is a Kmer graph from the banana slug genome.



Figure 1. Kmer graph from banana slug genome. Histograms from a variety of kmers are shown, as well as a fit of the main kmer peak for 15mers according to a gamma distribution. From: https://banana-slug.soe.ucsc.edu/bioinformatic\_tools:jellyfish) .

These graphs show multiplicity of a kmer (how many times is a kmer found in the read data), against number of distinct k-mer with the given multiplicity.

## Manual Exercise: Make your own kmer histogram and estimate genome size by hand

Given the following reads {GCA, CAT, ATG, TGC}:

* Make a kmer histogram based on the distribution of 2-mers from these reads, and estimate mean kmer size.
* Estimate genome size by summing the number of distinct kmers excluding error kmers (assume no errors occurred).
* Estimate genome size from the total number of non-error 2mers divided by mean 2mer coverage.
* Suppose there was an error in one of the reads (the shaded G in the kmer shown below) such that the read set was GGA, CAT, ATG, TGC. How many unique error kmers are created from the read with the error?

Errors cause many kmers to be affected, and cause a low frequency bump in kmer histograms with high genome coverage. For genomes, the shape of the kmer graph is fairly similar for kmers 15 or larger. Below that size many kmers are repeated many times throughout the genome. At 15 and larger there are two characteristic peaks as described above for errors and true coverage. Note that repetitive DNA will have higher coverage and causes a “heavy tail” for the main peak.

Goals

* Practice using the Linux Operating System
* Remote Cluster Computing
* Run an application in Linux
* Estimate genome size
* Estimate data error rate
* Develop and interpret Kmer graphs

V&C Competencies addressed

* Ability to apply the process of science: Evaluation of experimental evidence
* Ability to use quantitative reasoning: Developing and interpreting graphs, Applying statistical methods to diverse data, Managing and analyzing large data sets
* Use modeling and simulation to understand complex biological systems: Applying informatics tools, managing and analyzing large data sets

GCAT-SEEK sequencing requirements

Any

Computer/program requirements for data analysis

Linux OS, Jellyfish

Optional Cluster: Qsub

If starting from Window OS: Putty

If starting from Mac or Linux OS: SSH

Protocols

Log into the HHMI cluster as you did when practicing Unix.

## Use the cluster to run jellyfish

1. Log onto the cluster.
2. Make a directory entitled, jellyfish

$mkdir jellyfish

1. Move into jellyfish as your working directory

$cd jellyfish

1. Copy the raw fastq files for *S. aureus* from a shared directory into your jellyfish directory. After the file transfers, check the files and their size by using ls -l.

$cp /share/apps/sharedData/GCAT/1\_jellyfish/\* .

$ls –l

1. Concatenate all fastq files into a single file called, “input.fasta”

$cat \*.fastq > input.fastq

We will run the Jellyfish program using Qsub. Qsub submits jobs to the cluster queue. The Juniata College HHMI cluster consists of a headnode and four compute nodes. Jobs are submitted to a program called Qsub which then delegates the cluster resources and distributes the job across the compute nodes. Jobs (programs) which are not submitted to Qsub will run right on the headnode and will not take advantage of the processors on the compute nodes.

1. You have already copied a file called “jellyfish.qsub”. Open it for editing using nano. Then edit the highlighted section with your username in order to have the right working directory.

$nano jellyfish.qsub

#!/bin/bash

#PBS -k o

#PBS -l nodes=1:ppn=8

#PBS -l walltime=60:00

#PBS -N Saureus

#PBS -j oe

#================================================#

# Saureus #

#================================================#

# NODES AND PPN MUST MATCH ABOVE #

NODES=1

PPN=8

JOBSIZE=10000

module load jellyfish.1.11

workdir=/home/YOURUSERNAME/jellyfish

cd $workdir

jellyfish count -m 15 -o output -c 3 -s 20M -t 8 input.fastq **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. Send the job to the worker nodes by entering:

$qsub jellyfish.qsub

**Regarding Qsub**

* nodes: sets number of nodes you want to run on. HHMI has 4 nodes so this should be a number between 1 and 4.
* ppn: sets number of processors per node you want to use. The maximum number of processors per node that may be used is 32.
* N specifies *your job name*
* oe tells PBS to put both normal output and error output into the same output file.
* module: loads programs

### Regarding Running Jellyfish

-m 15 sets the size of kmers counted to 15

-o sets the prefix of output files

-c sets the number of bits to use for counting kmers in a single row

-s sets the size of the Hash table used to store and count kmers(20M = 20 million)

-t sets the number of threads (processors) the job is divided into

NOTE: -h or --usage after the command will explain the options available with most programs

H. To shows full listing of processes running in Qsub. Wait a few minutes until the job’s status changed from R (running) to C (complete).

$qstat -all

I. To delete your job if you make a mistake:

$qdel [job number]

J. Merge hash tables using the regular expression\\_\*. “\\_\* will find the \_ character and anything after it to merge into the output.jf file. Small jobs like this can be done with the head node.

$jellyfish merge –o output.jf output\\_\*

1. Check the results using:

$ls

1. Make a histogram of the Kmer counts. “-f” will include kmers with count zero (should be empty for Kmers, but wouldn’t be for qmers from Quake). “>” will send the results to a file named “histogram.txt.” Always be careful using “>”.

$jellyfish histo –f output.jf > histogram.txt

1. Check the histogram using $less. It should include two columns, with the first the multiplicity (X axis) and count (Y axis).

$less histogram.txt

1. Calculate stats on the qmers for estimate of error rate and genome size into a file called “stats.txt, ” and open it up.

$jellyfish stats –v –o stats.txt output.jf

$less stats.txt

1. Note that “unique” kmers are those represented only once in the data. “Distinct” kmers refers to the number of kmers not including their multiplicity (i.e. not summing across how many times they are found; i.e. how many kmers with different sequences were found including unique and non-unique kmers). Total kmers refers to the number of distinct kmers including their multiplicity [i.e. sum of (the number of kmers found at each multiplicity \* their multiplicity)].

So if there were 100 unique kmers, 30 found 2X, 40 found 3X, and 30 for 4X, the number of distinct kmers would be 200, the number of unique kmers is 100, the maximum number (max\_count) of kmers is 100, and total number of kmers is 400.

Assessment

1. Record frequency of the first 10 multiplicities below from the histogram file.

|  |  |
| --- | --- |
| **Multiplicity** | **Frequency** |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| 6 |  |
| 7 |  |
| 8 |  |
| 9 |  |
| 10 |  |

1. Counts typically start high indicating errors, dip, and then increase to designate true coverage, with a peak at the average kmer coverage. Identify the low point of the dip.
2. Take the sum of the kmers with multiplicities below the minimum above:\_\_\_\_\_\_\_\_\_\_. These represent mainly error reads.
3. What was the number of distinct kmers:\_\_\_\_\_\_\_\_
4. Subtract number of error kmers from number of distinct kmers:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_. **This is an estimate of genome size.**
5. Divide number of error kmers by number of distinct kmers x 100. **This is the percent of error kmers.** Divide further by kmer size to get an estimate of per bp error rate:\_\_\_\_\_\_\_\_\_\_
6. What is the total number of kmers:\_\_\_\_\_\_\_\_\_\_\_\_\_\_
7. At what multiplicity is the second kmer peak?\_\_\_\_\_\_\_\_\_\_\_\_. **This is an estimate of average kmer coverage.**
8. Divide total # kmers by average kmer coverage to get **another estimate of total genome size**:\_\_\_\_\_\_\_\_\_\_\_\_

Time line of module

One to two hours of lab

Discussion topics for class

Question: How close are your two estimates of genome size? Why might they differ?

References

### Literature Cited

Kenney DR, Schatz MC, Salzberg SL. 2010. Quake:quality-aware detection and correction of sequencing errors. Genome Biology 11:R116.

Marcais G, Kingsford C. 2011. A fast, lock-free approach for efficient parallel counting of occurrences of k-mers. Bioinformatics. 27:764-770. [Jellyfish program]

Module 2. Genome assembly II: Data Quality and Error correction

Background

Quality filtering and error correction has a profound influence on the quality of an assembly (Kenny et al. 200; Salzberg et al. 2012). While one might want to use highly stringent criteria for quality filtering, this can reduce the amount of data to the point that coverage is too low for assembly, and good data is thrown out. Conversely, including data of poor quality will cause many kmers that, if unique, will not assemble or if they mutate to an existing sequence, may cause misjoins. Many assembly programs have built in error correction algorithms before de Bruijn Graphs are contructed, like SOAPdenovo (Li et al. 2010; Luo et al 2012) and Allpaths-LG (Butler et al. 2008), but others do not, such as Velvet (Zerbino and Birney 2008).

Each entry of a Fastq files takes up 4 lines and looks like this:

@SRR022868.1209/1

TATAACTATTACTTTCTTTAAAAAATGTTTTTCGTATTTCTATTGATTCTGGTACATCTAAATCTATCCATG

+

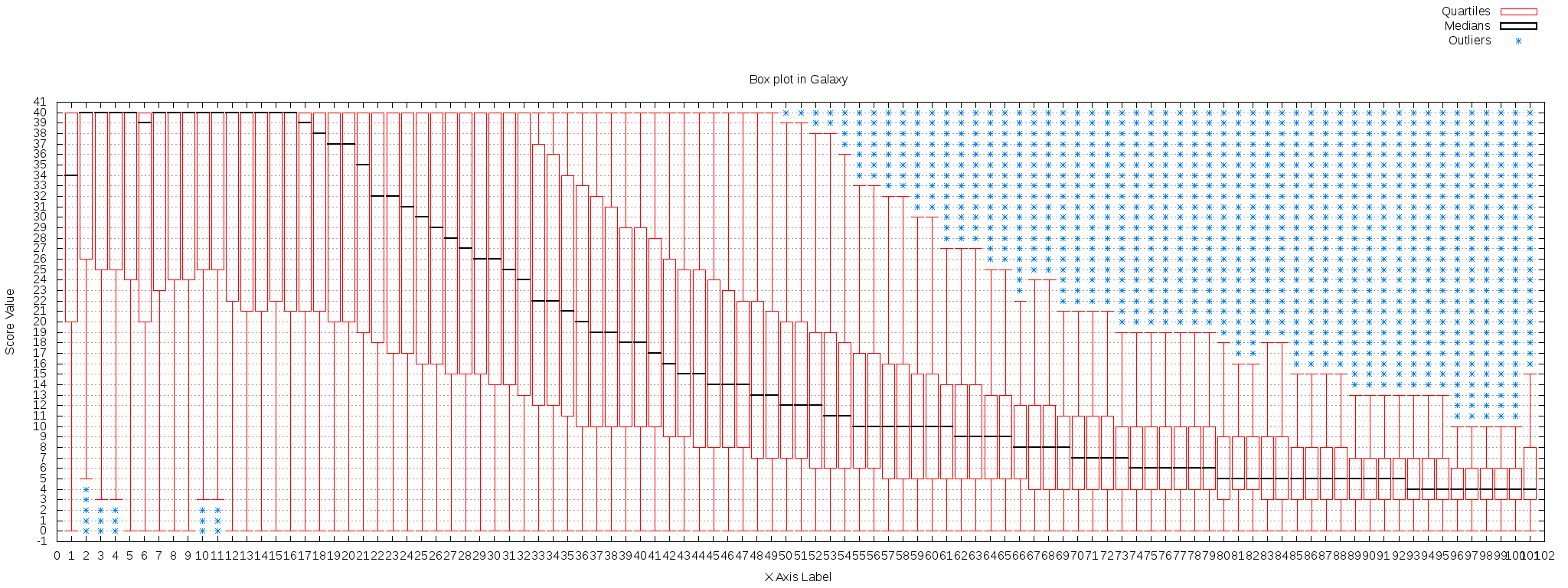
Unfortunately, there are a few other Phred score encodings depending on which version of the Illumina platform that data was sequenced on. For example, there are two formats where one has to subtract 64 instead of 33. Data in the NCBI archive has been converted to Sanger format. See the following link for all the fascinating details: http://en.wikipedia.org/wiki/FASTQ\_format.

III<IIIIIIIHIIIIIIIIIIII<IIBIIDIII;IIIIGI=IIII<C;III,AE.A<D.00E,?1B7,HFE

* The first line is the fastq header with the name of the read, ending with /1 or /2 for paired end or mate paired data. Its counterpart can be found in the same order in the paired library. Sometimes the files are concatenated into a single paired end data file. Note that most programs expect the pairs to be in the same order in two files, like they are in this example.
* The second line is the sequence. While N’s are used, other IUPAC ambiguity symbols are not.
* The third line is a separator between the sequence and quality scores that may include some optional text.
* The fourth line is comprised of “Phred 33” or “Sanger” quality scores. There is one score for each nucleotide. These are calculated as -10 \* log10 (probability of error). A nucleotide with an error probability of 0.01 would have a score of 20.

In order for the quality scores to take only one character of space, they are encoded as ASCII text, each character of which has a numerical, as well as text value. You can find a table with the values at <http://en.wikipedia.org/wiki/ASCII>. Note that the integer 33 is represented by “!” and 73 by “I”. To convert to numerical error probabilities, one would take the decimal value of the ASCII character (e.g. 73), subtract 33 (73-33=40), divide by -10 (40/-10=-4), and raise 10 to that value (e.g. 10^-4= 0.0001).

The following boxplot shows Phred quality scores (see above) on the y-axis and position within reads starting from the 5’ end (for all read simultaneously). Boxplots summarize quality data across the entire data set for each position. A decline in quality is shown from 5’ to 3’ position. The center of the box is the median, lower end of the box is the 25th percentile (Q1), top of the box the 75th percentile (Q3), whiskers show the data at about two standard deviations from the median [>1.5x(Q3-Q1) distant from Q1 or Q3] and blue stars are outliers.



*Reflection Question: Does this look like a good run?*

*Do you think one could assemble a genome with data of this quality?*

Several programs correct errors in reads using kmer graphs in whole genome sequencing projects. To use the graphs they identify error reads as those below the minimum point between the true and error kmer peaks. Each putative error kmer is examined for poor quality nucleotides and replacements are selected based on how their probability, as well as the common errors made by the sequencing platform.

In order to better separate peaks representing error and true kmers, rather than count each kmer one time for it’s occurrence the program Quake weights the kmer count by the product of [1-P(error)] over all nucleotides in the kmer. If all nucleotides in a kmer have very low error probabilities, the qmer count will be close to the number of kmers. For example, if 15 nucleotides in a kmer all had error probabilities of 0.01, the qmer count for that kmer would be 0.99^15 = 0.86. On the other hand, a kmer with poor quality (e.g. p(error) = 0.2 for each of 15 nucleotides) would result in a qmer count much less than one. For example 0.8^15=0.035. Five such nucleotides would still have a count of less than one (0.035\*5=0.18). Overall, conversion to qmer counts helps to separate true low coverage kmers from error kmers. Quake will filter out reads that cannot be corrected and still produce low qmers, and will trim poor quality nucleotides off read ends as well. This step is relatively computationally intensive. We we will use the HHMI cluster to do this job. In the example below we will see the effects of such error correction on files containing the *S. aureus* genome.

Goals

* Understand Phred quality scores and construct box plots
* Understand how kmer gaphs can be used to identify errors in reads
* File transfer from HHMI cluster to another cluster (Galaxy)
* Optional: Use Galaxy platform

V&C core competencies addressed

1) Ability to apply the process of science: Evaluation of experimental evidence

2) Ability to use quantitative reasoning: Developing and interpreting graphs, Applying statistical

methods to diverse data, Mathematical modeling, Managing and analyzing large data sets

3) Use modeling and simulation to understand complex biological systems:

Applying informatics tools, Managing and analyzing large data sets, Incorporating stochasticity into biological models

GCAT-SEEK sequencing requirements

Any

Computer/program requirements for data analysis

Linux OS, Quake

Optional Cluster: Qsub

If starting from Window OS: Putty

If starting from Mac OS: SSH

Protocols

1. Log onto the cluster. If your are already logged in, use “$cd “ to move to your home directory.
2. Make a directory entitled “quake” and enter it.

$mkdir quake

$cd quake

1. Move your four raw fastq files for *S. aureus* from the jellyfish directory into your current directory. We could have copied them but proliferating files takes up lots of space.

$mv ../jellyfish/\*1.fastq .

$mv ../jellyfish/\*2.fastq .

1. Copy the qsub file from a shared quake directory into the current directory.

$cp /share/apps/sharedData/GCAT/2\_quake/quake.qsub .

1. Make a new file and name it at the same time:

$nano fastqfiles

1. List the file names as below. Note that paired reads need to be on the same line of this file. Once you’ve typed the information below hit control-X and you will be prompted for save information.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

frag\_1.fastq frag\_2.fastq

shortjump\_1.fastq shortjump\_2.fastq

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

# Another Qsub Example

We will be using Qsub to submit our Quake job to the cluster queue. Edit the file quake.qsub to run quake.py on the cluster like before, using 16 processors on 1 node. Recall that you will need to edit your working directory

#!/bin/bash

#PBS -k o

#PBS -l nodes=1:ppn=16

#PBS -l walltime=60:00

#PBS -N Quake

#PBS -j oe

#================================================#

# Quake #

#================================================#

# NODES AND PPN MUST MATCH ABOVE #

NODES=1

PPN=16

JOBSIZE=10000

module load jellyfish.1.11

module load RMod

module load Quake

workdir=/home/YOURUSERNAME/quake

cd $workdir

quake.py -f fastqfiles -k 15 -p 16

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

The last line of the file will run quake.py script on 16 (-p 16) processors for a kmer of 15 (-k 15) on all four data sets at once, maintaining order of paired reads.

Send the job to the worker nodes by entering:

$qsub quake.qsub

It may take 10 minutes.

To shows full listing of processes running in Qsub:

$qstat -all

To check on any output from the program go to your home directory and nano the file

$less Quake.o[your job ID number]

To delete your job if you make a mistake:

$qdel [your job ID number]

1. Corrected data is in the fastq files with “cor” in the name. 1 bp takes up one Byte of storage space, so excluding headers, the size in MB = size in BP. Compare the size of the corrected and uncorrected datasets using:

$ls –la

1. Check the number of corrections in the first file:

$nano frag\_1.stats.txt

Assessment

Examine the cutoff.txt file. Compare the cutoff in the qmer analysis to the histogram low point for the same data in Jellyfish. What does it say about the data for the qmer cutoff to be shifted and lower?

Draw a theoretical qmer plot and label error peaks, true peaks, repeats.

What is the qmer score of 10 nucleotides with a phred score of 10 each.

Timeline of module

2 hours

Discussion topics for class

Describe the differences in the phred plots for quake corrected or uncorrected data.

What would the Phred score be for an error probability of 0.001? of 0.05?

What is the error probability for an ASCI quality score of “!”?

Why does the qmer histogram start with zero, whereas the kmer histogram started with 1?

References

### Literature Cited

Butler J, MacCallum I, Kleber M et al. 2008. ALLPATHS: De novo assembly of whole-genome shotgun microreads. Genome Res. 18: 810-820.

Kenney DR, Schatz MC, Salzberg SL. 2010. Quake:quality-aware detection and correction of sequencing errors. Genome Biology 11:R116

Li R, Zhu H, Ruan J et al. 2010. De novo assembly of human genomes with massively parallel short read sequenceing. Genom Res. 20:265-272 [SOAPdenovo 1]

Luo R, Liu B, Xie Y, et al. 2012. SOAPdenovo2: an empirically improved memory-efficient short-read de novo assembler. GigaScience 1:18. [important online supplements]

Salzberg SL, Phillippy AM, Zimin A, et al. 2012. GAGE: a critical evaluation of genome assemblies and assembly algorithms. Genome Res 22: 557-567. [important online supplements!]

Zerbino DR, Birney E. 2008. Velvet: Algorithms for de novo short read assembly using de Bruijn graphs. Genome Research. 18:821-829.

### Further Reading

Haddock SHD, Dunn CW. 2011. Practical computing for biologists. Sinauer Associates

Module 3. Genome assembly III: Assembly algorithms

Background

Modern genome sequencing involves construction of the millions of bp that makes up eukaryotic chromosomes, from millions of short random fragments of DNA about 100bp long that blanket the chromosomes. Large fragments can be constructed from small fragments if the fragments sufficiently overlap each other. A new analytical technique called De Bruijn graphing was developed to deal with the difficult computational problem of comparison all DNA reads against every other read, when millions of reads are in a single dataset . The method involves decomposing raw DNA reads into short stretches of DNA called Kmers, and extension of DNA chains from exact matches of Kmers differing by a single nucleotide from each other.

Consider the small circular genome:

1. Say the following set of 3bp reads are obtained from randomly sequencing the genome {ATG, CAT, TGC, GCA}. To construct a genome using a De Bruijn graph:

**G**

**C**

**A**

**T**

1. Break the reads into smaller segments called kmers. A kmer of size 2 will produce the following dimers: {AT, TG, CA, AT, TG, GC, GC, CA}.

AT

TG

**T**

**G**

**C**

**A**

**T**

**G**

**C**

**A**

CA

GC

3. Start the graph by making vertices (bubbles) that contain prefixes (first k-1 letters of the kmers) or suffixes (last k-1 letters of the kmers). Write each unique prefix or suffix just once. In this case it is simply the letters A,G,C, & T.

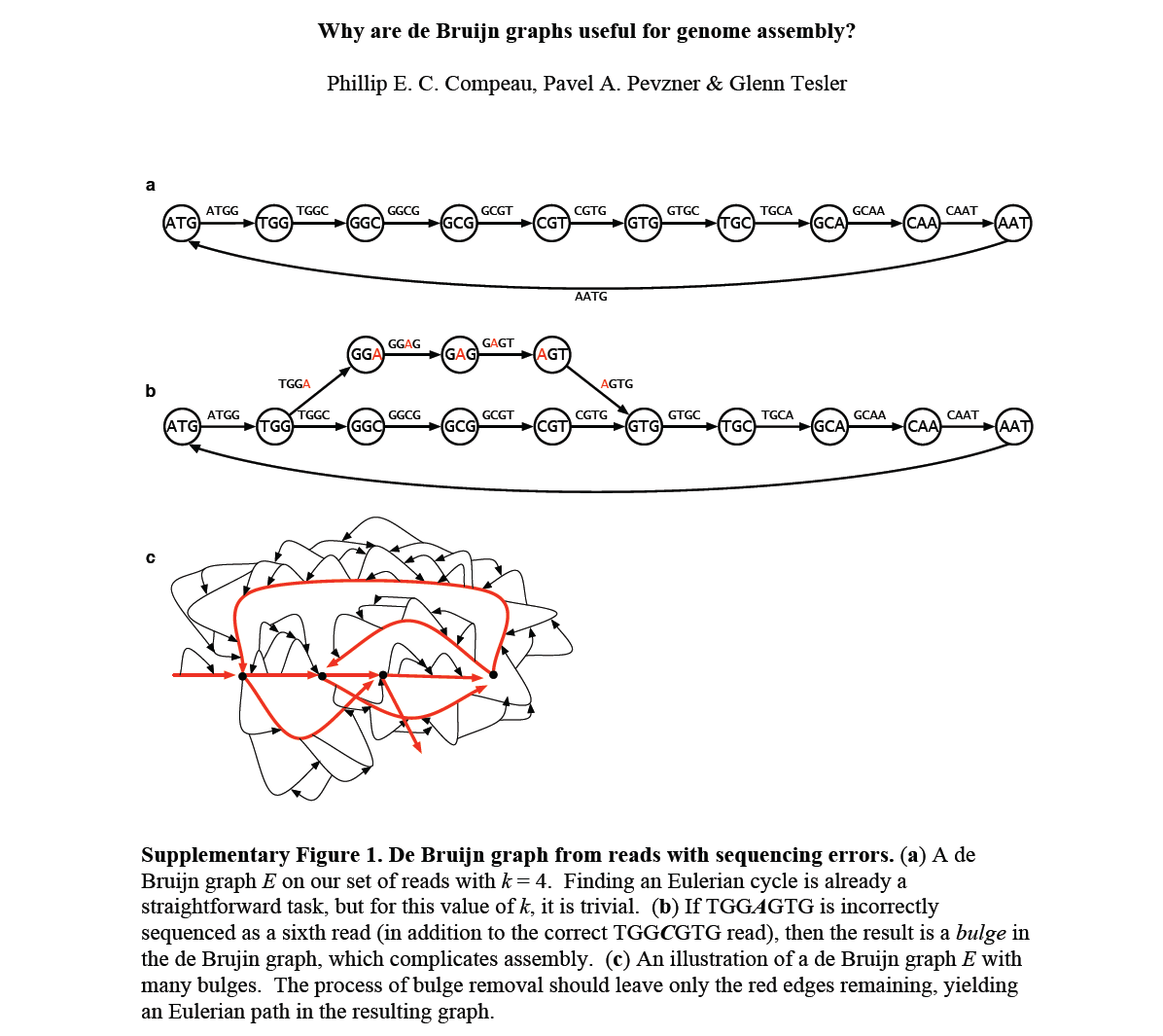
4. Connect the vertices using numbered, directed arrows (edges) labeled with kmers that contain both the prefix and suffix. Consider how each kmer connects vertices. The graph is done when all kmers are considered. To construct the genome sequence, connect the last letter of each kmer in the order determined by the graph. Use each edge (kmer) only once. Note that not all kmers exist (e.g. AG), since that isn’t part of the genome sequence.

**Q1. Now you try**. Say the following set of 4bp reads are obtained from randomly sequencing a different genome {GCAT, GGCA, CATT, ATTA, TTAG, AGGC, TAGG}. Construct the genome sequence using a DeBruijn graphing approach, using kmer size of 3, following the 4 steps above!

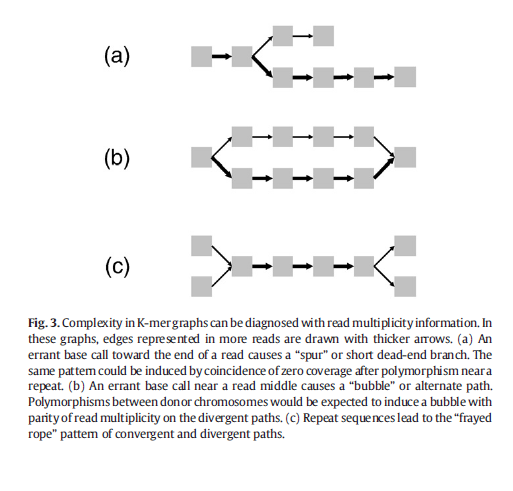
**Q2. Fragmentation.** The problem with genome construction from random (aka shotgun) short read data is that despite high coverage of the genome, assemblies are fragmented. What would your assembly above look like if you only had reads {GGCA, ATTA, AGGC}? How many contigs would result? What would the sequence be?

**Q3. Repeats.** Another major problem with genome construction is that repetitive DNA sequences make it impossible to reconstruct genomes from short reads. The resulting graphs are “tangled”. Try reconstructing the circular genome (CTATCATCG) using the DeBruijn graphing approach, from the following set of 3bp reads {CTA, TAT, ATC, TCA, CAT, TCG, CGC, GCT}.

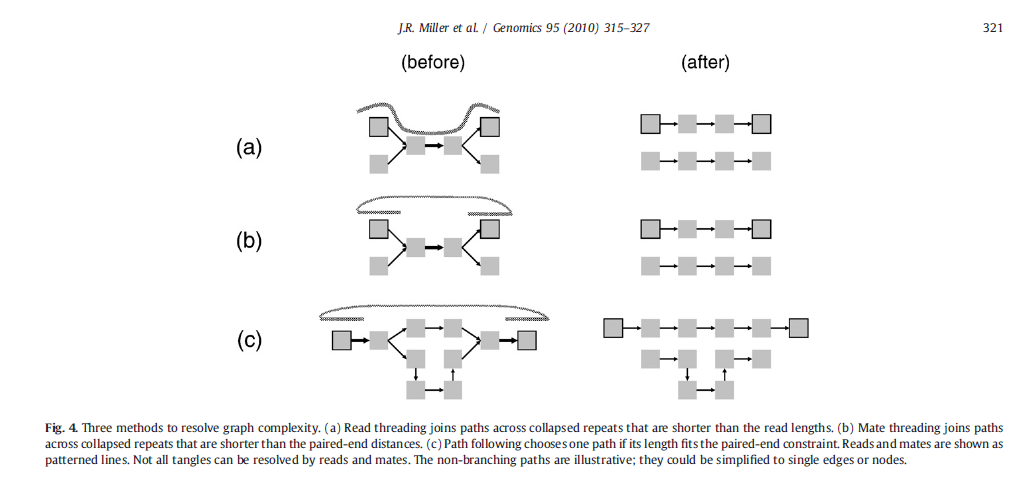
**Errors**. Errors in sequence reads greatly complicate graph construction. Each error causes many kmers to be affected, and cause “bulges” in de Bruijn graphs. The figure below illustrates a read error in red, and shows how if kmer size is 4, each error will affect 4 kmers.



The figure below (Millet et al. 2010) shows common problems with DeBruijn graphs. A “spur” shape is formed when the error occurs at the end of the read (A). A “bubble” shape is formed when an error occurs in the middle of a read (B). A “frayed rope” pattern occurs when a read contains repeats (C). Frequency of kmers (shown as thickness of arrows) is used by most genome assemblers to error-correct graphs.



The figure below (Miller et al. 2010) shows how reads spanning the length of a repeat (a), paired ends spanning either side of a repeat (b), and knowledge of distance between read pairs (c), can be used to resolve ambiguities in a graph.



**Key Points**

Repeats can be resolved if kmers or reads extend longer than the repeat length, or if insert size exceeds repeat length, and both ends of the insert are sequenced. However, large kmers may result in genome fragmentation (i.e. reduce contig size) because DeBruijn graphs extend DNA chains by one nucleotide at a time, and a single missing kmer will break extension.

# SOAPdenovo

SOAPdenovo (Li et al. 2010; Luo et al 2012) is comprised of 4 distinct commands that typically run at the same time.

Pregraph: construct kmer-graph

Contig: eliminate errors and output contigs

Map: map reads to contigs

Scaff: construct scaffolds

All: do all of the above in turn

Error correction in SOAPdenovo itself includes calculating kmer frequencies and filtering kmers below a certain frequency (in high coverage sequencing experiments, rare kmers are usually the result of sequencing error), and correcting bubbles and frayed robe patterns. It creates DeBruijn graphs and to create scaffolds, maps all paired reads to contig consensus sequences, including reads not used in the graph. Below we will try assembly with all SOAPdenovo modules, on data that has already been through error correction screens.

Goals

* *De novo* De Bruijn genome assembly by hand and using Linux OS
* Effects of key variables on assembly quality
* Measuring assembly quality

V&C core competencies addressed

1) Ability to apply the process of science: Observational strategies, Hypothesis testing, Experimental design, Evaluation of experimental evidence, Developing problem-solving strategies

2) Ability to use quantitative reasoning: Developing and interpreting graphs, Applying statistical

methods to diverse data, Mathematical modeling, Managing and analyzing large data sets

3) Use modeling and simulation to understand complex biological systems: Computational

modeling of dynamic systems, Applying informatics tools, Managing and analyzing large data sets, Incorporating stochasticity into biological models

GCAT-SEEK sequencing requirements

any

Computer/program requirements for data analysis

Linux OS, SOAPdenovo 2, Quast

If using a Cluster: Qsub

If starting from Window OS: Putty

If starting from Mac or Linux OS: SSH

Protocols

# Genome assembly using SOAPdenovo2 in the Linux environment

We will perform genome assembly on Juniata-HHMI cluster. In this environment CAPITALIZATION and PERFECT SPELLING REALLY REALLY COUNTS!!! Our work will focus on bacterial genomes for purposes of brevity, but this work, in this computing environment, can be applied to even mammalian sized genomes (>1Gbp). We will assemble from 2 error-corrected bacterial genome files. The bacterial genome size is ~ 5Mbp. The raw data includes a paired-end fragment library with an “insert length” of 550bp, in “innie” orientation (forward and reverse reads facing each other), with 2x250bp paired end MiSeq reads providing up to 100x genome coverage.

1. Go to the directory with your last name (using cd) and make a directory entitled, soap, check that it is there, and move into it.

$mkdir soap

$ls

$cd soap

1. Copy all the data from our GCAT shared directory to your directory. Don’t forget the space & “.” at the end of the line. Your may divide analysis of different datasets among team members. After copying the data, use ls -l to see what was copied, as well as how big the different files are.

$cp /share/apps/sharedData/GCAT/3\_soap/\* .

$ls -l

1. Edit the config.txt file using nano. This file tells SOAP which files to use, and where you will enter important characteristics of the data. Get started by entering

$nano config.txt

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#maximal read length

max\_rd\_len=250

#below starts a new library

[LIB]

#average insert size

avg\_ins=550

#if sequence needs to be reversed put 1, otherwise 0

reverse\_seq=0

#in which part(s) the reads are used. Flag of 1 means only contigs.

asm\_flags=1

#use only first 250 bps of each read

rd\_len\_cutoff=250

#in which order the reads are used while scaffolding. Small fragments usually first, but worth playing with.

rank=1

# cutoff of pair number for a reliable connection (at least 3 for short insert size). Not sure what this means.

pair\_num\_cutoff=3

#minimum aligned length to contigs for a reliable read location (at least 32 for short insert size)

map\_len=32

#a pair of fastq files, read 1 file should always be followed by read 2 file

q1=Ensifersp\_R1\_1X\_cor.fastq

q2=Ensifersp\_R2\_1X\_cor.fastq

Hit control-X to exit. Notice that the last two lines control the data input.

1. Run Soapdenovo using Qsub. First use $nano to edit the file soap.qsub and edit the working directory, as above. We will use the high memory (127mer) version of SOAPdenovo, a kmer (-K) of 21, 8 processors (-p), the config file you just made (-s), and give all output files the prefix ens1XK21 (-o). The last line of the Qsub control file specifies these parameters. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#!/bin/bash

#PBS -k o

#PBS -l nodes=1:ppn=8

#PBS -l walltime=60:00

#PBS -N Soap

#PBS -j oe

#================================================#

# Soap #

#================================================#

# NODES AND PPN MUST MATCH ABOVE #

NODES=1

PPN=8

JOBSIZE=10000

module load SOAPdenovo2

workdir=/home/YOURUSERNAME/soap

cd $workdir

SOAPdenovo-127mer all -K 21 -p 8 -s config.txt -o ens1XK21

**SOAPdenovo run details**

* SOAPdenovo-127mer allows kmer size to vary up to 127
* K sets kmer size (e.g. 21)
* p is the total cpus (must match #nodes \* #ppn; e.g. 1\*8=8)
* s identifies the configuration file
* sets a prefix for all the output files for that run

1. Run qsub using

$qsub soap.qsub

1. Examine directory contents using $ls
2. You will see a job identifier appear. Write the number down. For 1X and 10X datasets, the job will take about a minute. Check to see if it is running by typing:

$ qstat -all

To see if your job is running, find your job number and look for an “R” in status. When the job is complete, you will see a “C.” After a couple minutes, use “ls” to see if any output files have appeared. If not, there has probably been an input error. To see what went wrong, go back to the home directory using cd ~, use “ls” to see the exact name of files, and use “$less” to open up the “Soap\_output” file that ends with your job number. This file will tell you the details of your Qsub submission.

$ cd ~

$ ls

1. $ less SOAP\_output.your\_job\_number
2. Page down within the file to find the error message and try to troubleshoot using that information. To quit “less,” press “q”. Get back to your previous directory using

$ cd –

1. When SOAPdenovo finishes, examine the soap directory contents using $ls. Each successful run of SOAPdenovo generates about 20 files, only a few of which contain information that we will use.

$ls

$less the \*.scafSeq file. It will have the scaffold result files. Does it look like you expected?

$tail –n 50 the \*.scafSeq file. This will reveal the tail end of the file (last 50 lines), which contains the largest assembled sequences.

$less the \*.scafStatistics file. This will contain detailed information on scaffold assembly. Record the relevant information in the table below and compare with a group member.

Size\_includeN: Total size of assembly in scaffolds, including Ns

Size\_withoutN: Total size of assembly in scaffolds, not including Ns

Scaffold\_Num: Number of scaffolds

Mean\_Size: Mean size of scaffolds

Median\_Size: Median size of scaffolds

Longest\_Seq: Longest scaffold

Shortest\_Seq: Shortest scaffold

Singleton\_Num: Number of singletons

Average\_length\_of\_break(N)\_in\_scaffold: Average length of unknown nucleotides (N) in scaffolds

Also contained will be counts of scaffolds above certain sizes, percent of each nucleotide and N (Gap) values, and “N statistics.” Recall that an N50 is the size of the smallest scaffold such that 50% of the genome is contained in scaffolds of size N50 or larger (Salzberg et al. 2012).

A line here showing “N50 35836 28” indicates that there are 28 scaffolds of at least 35836 nucleotides, and that they contain at least 50% of the overall assembly length. Statistics for contigs (pre-scaffold assemblies) are also shown.

H. Rerun the analysis with the parameters in the table below.

* To change the kmer size, edit the final line of the qsub script (the SOAP line) by adjusting the –K option.
* Edit the config.txt file to change the names of the input files (they differ in being 1X, 10X, or 100X).
* Each time you change kmer size or input file name adjust the –o option in the Qsub Script to change the names of the output files. For example, use “ens1XK21” to label output files as resulting from the 1X data, with Kmer of 21.
* Run the new Qsub script.

**Fill in the table below and cut/paste into your notebook observations/results.**



If there is time…you can help “finish” a genome by mapping reads to resulting scaffolds and filling in overlapping sequence that didn’t make it into the original assembly. Do this by running GapCloser with Qsub once you have the optimal assembly. Use the same config file you made above (-b), fill in the sequence file \*.scafSeq (-a), make the output file \*.scafSeq, use 16 processers (-t 16), and make use there is an ovelap of 31 nt before gap filling (-p).

$GapCloser -b config.txt -a ens1xK21.scafSeq -o ens1xK21.scafSeq -t 16 -p 31

Assessment/Discussion

* Discuss effects of changing kmer size and coverage on quality of assembly. Support your arguments citing specific results.
* *For Later*: Assemble the *S. aureus* genome data from the earlier modules, with and without error correction.
* Look at the SOAP manual to see how to include both a regular paired-end and mate-pair library. Use the information from the original web page to determine library properties. http://gage.cbcb.umd.edu/data/Staphylococcus\_aureus/Data.original/readme.txt

Time line of module

Two hours

References

### Literature Cited

Compeau PEC, Pevzner PA, Tesler G. 2011. How to apply de Bruijn graphs to genome assembly. Nature Biotechnology. 29:987-991.

Margulies M, Egholm M, Altman WE, Attiya S, Bader JS, Bemben LA, Berka J, Braverman MS, Chen Y-J, Chen Z et al . 2005. Genome sequencing in open fabricated high density picoliter reactors. Nature 437: 376–380.

Miller JR, Koren S, Sutton G. 2010. Assembly algorithms for next-generation sequencing data. Genomics 95: 315-327.

Salzberg SL, Phillippy AM, Zimin A, et al. 2012. GAGE: a critical evaluation of genome assemblies and assembly algorithms. Genome Res 22: 557-567. [important online supplements!]

Module 4. Genome annotation with Maker I: Introduction and Repeat Finding

Background

The Maker Annotation Pipeline

“Maker” is a genome annotation pipeline (Cantarel et al. 2008; Holt and Yandell 2012; figure next page) that combines two kinds of information to make structural gene annotations from raw DNA: i) extrinsic “evidence”, which is evidence supplied to Maker based on similarity of genomic regions to other organisms’ mRNA (also known as expressed sequence tags, or ESTs) and protein sequences and ii) gene predictions from intrinsic signals in the organism’s DNA found by *ab-initio* (from scratch) gene predictors. Intrinsic evidence is evidence is evidence such as start and stop codons and intron-exon boundaries predicted from gene predictors. The two gene predictors used by Maker are SNAP and Augustus. Maker can also be run with a greater “weight” on EST or protein evidence, by allowing gene predictions without support from the *ab initio* gene predictors, which typically miss many genes. MAKER takes the entire pool of *ab initio* and evidence-informed gene predictions, updates features such as 5' and 3' UTRs based on EST evidence, tries to determine alternative splice forms where EST data permits, and chooses from among all the gene model possibilities the one that best matches the evidence. Annotations are only produced if they are supported by both BLAST alignments and a gene predictor.

**Assembled Genome**

**Repeat Masker**

**Masked Genome**

**Gene Predictions**

**Ab initio gene predictors**

**SNAP & Augustus**

**Training**

**Whole Genome**

**Est2Genome**

**Extrinsic**

**Evidence**

**Proteins**

**ESTs**

**MAKER Gene Models: best match between Extrinsic Evidence and Gene Predictions**

The first step of the **multi-step Maker gene annotation pipeline** involves finding repetitive DNA and labeling (i.e. “masking”) the genomic DNA using the program RepeatMasker. Repeat masking is important in order to prevent inserted viral exons from being counted as fish exons. The *ab-initio* gene predictor “SNAP” can detect the intrinsic signals in the DNA model genes. Because “what a gene looks like” differs significantly among genomes, gene finders must be “trained” to know what the signals look like in each new species sequenced. The SNAP training file (Sru.hmm) was prepared by your instructors from a training set of high quality (complete) genes obtained from the longest scaffolds in the dataset, but one would start with the most closely related species available. Maker uses BLAST (Basic local alignment search tool) to align protein and mRNA sequences to your genome sequence.

### Gene annotations are stored in GFF3 file format

General Feature Format (GFF3) is a standard file format, meaning that it is supported and used by several programs and appears in the same form regardless of what program generated it. GFF is a tab delimited file featuring nine columns of sequence data.

|  |  |  |
| --- | --- | --- |
| Column | Feature | Description |
| 1 | seqname | The name of the sequence |
| 2 | source | The program that created the feature |
| 3 | feature | What the sequence is (gene, contig, exon, match) |
| 4 | start | First base in the sequence |
| 5 | end | Last base in the sequence |
| 6 | Score | Score of the feature |
| 7 | strand | + or - |
| 8 | frame | 0,1, or 2 first, second, or thrid frame |
| 9 | attribute | List of miscellaneous features not covered in the first 8 columns, each feature separated by a ; |

Example gff (Shows part of a single exon maker gene found on scaffold 1)



A large fraction of the eukaryotic genome is in the form of dispersed repeats. Short and Long Interspersed Nuclear Elements (SINES or LINES) are typically around 350bp or 6Kbp, respectively. They may contain coding genes from exogenous sources such as retroviruses. These genes need to be identified as exogenous to keep gene finders from confusing them with endogenous genes necessary for the function of the organism. To avoid confusing gene families with repeats, members of gene families are identified by Blast search against databases, and removed from the repeat library manually.

Custom repeat libraries are combined with known repeats from other organisms for comprehensive repeat identification. Once repeats are identified, the genome is searched for these repeats and matching sections are “masked” by converting to lower case (soft masked) or to N (hard masked) in order to prevent gene finders from calling them genes.

# RepeatScout

RepeatScout (Price et al. 2005) assembles a custom repeat library for use in RepeatMasker. RepeatScout counts every 12 base pair sequence, then extends those 12 base pair sequences to form consensus sequences. Simple and tandem repeats are then filtered from the preliminary repeat library. The filtered repeat library is then used in RepeatMasker to mask the genome. Repeats that are found less than ten times within the entire genome are then also removed. To ensure that conserved protein families are not miss-labeled as repeats, BLAST2GO (Conesa et al. 2005) is used to determine similarity between the putative repeats and any known proteins. Finally, repeats are classified using TEclass (Abrujan et al. 2009).

Goals

* Find and characterize repetitive elements specific to the novel genome to be masked in gene annotation.

V&C core competencies addressed

Ability to use quantitative reasoning: Applying statistical methods to diverse data, Mathematical modeling, Managing and analyzing large data sets

GCAT-SEEK sequencing requirements

None

Computer/program requirements for data analysis

Linux OS, repeat masker, repeat scout, TE class on web browser

Protocols

# Repeat Scout

RepeatScout is run in several steps using several different programs. This tutorial assumes you are using a fasta file named genome.fa you could, however, use any fasta file simple replacing genome.fa with the name of the fasta file being used. Also, running any of the listed commands without any additional input (or running them with the -h option) will display the help message including a description of the program, the usage, and all options.

1) From your home directory, make a new directory called repeatScout using, and copy three large scaffolds from test fish genome and a base qsub file (repeat1.qsub) into the folder…

$mkdir repeatScout

$cp /share/apps/sharedData/GCAT/4\_repeatScout/\* .

2) The provided Qsub script will count every 12 base pair sequence in the genome. The Qsub script will load RepeatScout, RepeatMasker, and perl modules. Make sure to edit your working directory path (highlighted below). Run the qsub script like normal, and when it is done, look at the result file using less.

#!/bin/bash

#PBS -k o

#PBS -l nodes=1:ppn=1

#PBS -l walltime=60:00

#PBS -N rptAnalysis

#PBS -j oe

#================================================#

# rptAnalysis #

#================================================#

# NODES AND PPN MUST MATCH ABOVE #

NODES=1

PPN=1

JOBSIZE=10000

module load RepeatMasker

module load RepeatScout

module load perl-5.14.2

workdir=/home/USERNAME/repeatScout/

cd $workdir

build\_lmer\_table -sequence genome.fa -freq genome.freq

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

$qsub repeat1.qsub

$less genome.freq

3) Edit the last line of the qsub script to extend the 12 base pair sequences to form initial consensus repeats. This will take a minute. (Note that if you have a large >50Mbp genome, you may want to increase run time (walltime) to 1000 minutes).

$RepeatScout -sequence genome.fa -freq genome.freq -output genome.firstrepeats.lib

4) Filter out simple and tandem repeats from the library (just use the headnode for this quick step)

$module load RepeatMasker

$module load nseg

$cat genome.firstrepeats.lib | filter-stage-1.prl > genome.filtered.lib

5) Edit the qsub script to count the number of times each repeat in the filtered library appears. The output file genome.fa.out will contain a column with the count of each repeat type. Use 8 processors.

$RepeatMasker -pa 8 -lib genome.filtered.lib -nolow genome.fa

\*The -pa option tells RepeatMasker how many processors to use. If you are only using one processor, do not use the -pa option. If you were running RepeatMasker on 3 processors, you would use -pa 3.

\*\*The -nolow option stops RepeatMasker from masking low complexity repeats. Since we are only concerned with the number of times the generated repeats appear, masking low complexity repeats simply adds more time.

6) Filter out any repeats that appear less than 10 times. Note, because our sample data is so small, not much will be repeated 10 times, so check the results when the first command finishes.

$cat genome.filtered.lib | filter-stage-2.prl --cat genome.fa.out > genome.secondfilter.lib

$less genome.secondfilter.lib

7) Here you can optionally use use TE class (below) to label each repeat type and for “unclear” repeats, use NCBI BlastX or Blast2GO to pull out members of legitimate gene families from genome.secondfilter.lib.

Edit the qsub script again to mask the genome file with the new repeat library.

$RepeatMasker -pa [number of processors] -lib genome.secondfilter.lib genome.fa

8) To view the length and frequency of repeats, open genome.fa.tbl. Total length, %GC, and percent bases masked (i.e. percent in repeats) are shown. Also shown are number and total length occupied by different kinds of repeats.

# TEclass

TEclass (Abrusan et al. 2009) will label unknown eukaryotic transposable repeat elements in the fasta repeat library by their mode of transposition (DNA transposons, LINES, SINES, LTRs; <http://www.bioinformatics.uni-muenster.de/tools/teclass/?lang=en&mscl=0&cscl=0>). The wiki description of transposon repeat classification is at: <http://en.wikipedia.org/wiki/Transposon>

We analyze repeats in different size categories: 0-600 bp, 601-1800 bp, 1801-4000 bp, >4000 bp, and built independent classifiers for all these length classes. We use libsvm as the SVM engine, with a Gaussian kernel. The classification process is binary, with the following steps: forward versus reverse sequence orientation > DNA versus Retrotransposon > LTRs versus nonLTRs (for retroelements) > LINEs versus SINEs (for nonLTR repeats). The last step is performed only for repeats with lengths below 1800 bp, because we are not aware of SINEs longer than 1800 bp. Separate classifiers were built for each length class and for each classification step. If the different methods of classification lead to conflicting results, TEclass reports the repeat either as unknown, or as the last category where the clasification methods are in agreement.

1) Go to <http://www.bioinformatics.uni-muenster.de/tools/teclass/?lang=en&mscl=0&cscl=0>

2) FTP or mail genome.secondfilter.lib to your laptop. Click “choose file” and upload genome.secondfilter.lib

3) Click run.

4) Results will be downloadable as the fasta file “input.lib”. Download and copy back into the Unix OS: to get the data to Unix, you can use FTP for the file, but since it is a small amount of information, it is easier to open a new document in the Unix OS using nano, return to windows and use Control-A (command-A for mac) to select all, Control-C to copy, and hit right click back into the nano document to paste, then save as usual. Results are comprised of a fasta file of repeats labeled with their category, named “input.lib”. Use >grep to pull out “unclear” repeats for Blastx analysis against teleost proteins using NCBI (Blast2go is slow, but is explained below FYI). The command below finds “unclear” in a fasta header and grabs the line under it too using “-A1”.

$grep –A1 “unclear” input.lib > unclear

5) Transfer the “unclear” repeats file back to your desktop, go to the NCBI BlastX page, upload the unclear repeats, and adjust parameters. Search against the “nr” database and restrict organism to “teleostei.” When finished, there will be a results pull down box showing which repeats had significant hits or not. Once you’ve identified repeats from real genes, write down the names, go back to Unix and delete them from input.lib using nano.

6) In order for RepeatMasker to read the identity of the repeats from TEclass we will need to edit the fasta header and move pieces around using special Find/Replace techniques.

We need to go from:

test.lib

>R=1 (RR=2. TRF=0.000 NSEG=0.000)|TEclass result: unclear

TTAGGTTTAGGCAACAAAACTACTTAGTTAGGTTTAGGAAAAAATCATGGTTTGGGTTAAAATAACT

>R=10 (RR=10. TRF=0.175 NSEG=0.000)|TEclass result: DNA

TTATTACACGGCTTTGTTGAATACTCGATTCTGATTGGTCAATCACGGCGTTCTACGGTCTGTTA

To:

test.lib

>R=1#unclear

TTAGGTTTAGGCAACAAAACTACTTAGTTAGGTTTAGGAAAAAATCATGGTTTGGGTTAAAATAACT

>R=10#DNA

TTATTACACGGCTTTGTTGAATACTCGATTCTGATTGGTCAATCACGGCGTTCTACGGTCTGTTA

We will use a “perl pie” to make the edits from the Unix command line. In general they work like this:

$perl –p –i.bak –e ‘s/find/replace/g’ filename

-p and –e tell perl that this is an executable process. –i.bak creates a backup file

* Find and replace use regular expression degeneracies:

\d represents digits, \s whitespace, \w words, + represents more than one of the preceding type of character.

* Find captures text in () and replaces the found values using the order in which they were detected with \1\2 etc.

$module load perl-5.14.2

$perl –p –i.bak –e ‘s/^(>R=\d+)(.+:\s+)(\w+)/\1#\3/g’ test.lib

Below you see how different segments of the header were captured by \1, \2, and \3. Note that we are deleting \2 by not putting it in the “replace” section.

{>R=1}{ (RR=2. TRF=0.000 NSEG=0.000)|TEclass result: }{unclear}

\1 \2 \3

7. Make sure it works on the test file, convert your TEclass library file, then rename it genome.secondfilter.lib ($mv input.lib genome.secondfilter.lib) and use it for step (7) in RepeatScout above.

# For You Reference: BLAST2GO

The point of this step is to remove uncharacterized repeats in case they represent motifs from multigene families like olfactory receptors. Again, you could do this after TE class and analyze ‘unclear’ repeats, but a batch NCBI search is faster.

1) Go to <http://www.blast2go.com/b2ghome>

2) Select the memory size you wish to use, then click “click here”

3) Open or run the downloaded .jnlp file.

4) Go to file -> load sequences and navigate to the folder with unclear repeats.

5) Go to blast -> run blast step

6) A window will appear with several blast options. Set the blast program to blastx. Set the blast database and blast expect value.

7) Click the play button and set a location for the blast output.

8) Download the results table and open in Excel or some other spreadsheet.

9) Search through the seq description column for any sequences that had similarities to known proteins.

10) Once a list of sequences with similarity to known proteins is made, run the command:

nano genome.secondfilter.lib or open genome.seconfilter.lib in any text editor that has a search feature.

11) Search for each sequence and remove those sequnces (ctrl-W) in nano.

Assessment/Discussion Questions

Was the average length of LINES as long as expected? Hint: fragment length of the rockfish libraries was about 350bp, with one PE sequencing library, and one MP library only.

References

### Literature Cited

Abrusan G, Grundmann N, DeMeester L, Makalowski W. 2009. TEclass: a tool for automated classification of unknown eukaryotic transposable elements. Bioinformatics 25:1329-1330

Cantarel B, Korf I, Robb SMC, Parra G, Ross E, Moore B, Holt C, Sanchez Alvarado A, Yandell M. 2008.MAKER: An Easy-to-use Annotation Pipeline Designed for Emerging Model Organism Genomes. Genome Research 2008 18: 188-96.

Holt C, Yandell M. 2011. MAKER2: an annotation pipeline and genome-database management tool for second-generation genome projects. BMC Bioinfo 12:491

Price AL, Jones NC, Pevzner PA. 2005. De novo identification of repeat families in large genomes.Bioinformatics. 21:I351–I358 [repeat scout]

### BLAST2GO

Conesa A, et al. 2005. Blast2GO: a universal tool for annotation, visualization and analysis in functional genomics research, Bioinformatics 21: 3674-3676.

Conesa A and Götz S. 2008. Blast2GO: A Comprehensive Suite for Functional Analysis in Plant Genomics. International Journal of Plant Genomics: 2008: 1-13.

Götz S et al. 2008. High-throughput functional annotation and data mining with the Blast2GO suite. Nucleic Acids Research 36:3420-3435.

Götz S et al. 2011. B2G-FAR, a species centered GO annotation repository. Bioinformatics 27: 919-924.

Module 5. Genome annotation with Maker II. Whole Genome Analysis

Background

Finding and characterizing genes in eukaryotic genomes is still a difficult and unsolved problem. Genomes differ enough from one another that, unless a gene finder from a closely related species is available, the most accurate gene finding necessitates training of gene finding algorithms. The major steps of annotation covered in this chapter are: i) train Maker to find genes in the genome of interest focused on long scaffolds likely to contain full genes, ii) run Maker on a whole genome, iii) analyze the gene annotation results.

Training. Maker is trained by doing a Maker run that generates a refined gene model more closely suited for the genome of interest than what is typically available. Gene finder training takes several hundred complete gene models and we are only going to *discuss* how to do it here. One generates the training set by first selecting the longest scaffolds in the assembly (total length likely to contain ~500 genes), then performing protein and est blast searches against those scaffolds in Maker, and then using est2genome and protein2genome options in the file “maker\_opts” to generate a collection of gene models.

Two primary gene finders that Maker relies on are SNAP (Korf 2004) and Augustus (Stanke & Waack 2003; Stanke et al. 2006). SNAP comes installed with MAKER and itself contains “fathom” and “forge”. These programs generate a “hidden-markov model” or HMM file that summarize information regarding the frequency of nucleotides at various sites within a single typical gene. Augustus has its own web-based training protocol that uses as input, the training annotations.

We will return to training after showing how to run the program on a full data set.

V&C core competencies addressed

1) Ability to use quantitative reasoning: Developing and interpreting graphs, Applying statistical

methods to diverse data, Managing and analyzing large data sets

2) Use modeling and simulation to understand complex biological systems: Applying informatics tools, Managing and analyzing large data sets

* Generate gene training files for a novel genome
* Run Maker in parallel on a high performance cluster for a whole genome

GCAT-SEEK sequencing requirements

None

Computer/program requirements for data analysis

Linux OS

Optional Cluster: Qsub

If starting from Window OS: Putty

If starting from Mac or Linux OS: SSH

Protocols

# I) Training (see appendix)

You will want to do this before the full Maker run, but it is easier to learn the standard run first.

# II) Run Maker

This tutorial assumes you are using a fasta file named testScaf500.fa you could, however, use any fasta file simple replacing testScaf500.fa with the name of the fasta file being used. This also assumes that you are using a custom repeat library created in the RepeatScout tutorial, a protein file named proteins.fa, a snap HMM file with an “.hmm” extension, and an Augustus configuration directory. Running any of the listed commands without any additional input (or running them with the -h option) will display the help message including a description of the program, the usage, and all options.

1. First , make a new folder from your home directory, move into it, and then copy the Maker control and other sample files from our shared directory into your directory.

$mkdir maker

$cd maker

$cp /share/apps/sharedData/GCAT/5\_maker/\* .

2) The maker\_opts.ctl file contains all of the performance options for maker. The lines highlighted below are those that are typically varied.

### maker\_opts.ctl

#-----Genome (these are always required)

genome= testScaf500.fa #genome sequence (fasta file or fasta embeded in GFF3 file)

organism\_type=eukaryotic #eukaryotic or prokaryotic. Default is eukaryotic

#-----Re-annotation Using MAKER Derived GFF3

maker\_gff= #MAKER derived GFF3 file

est\_pass=0 #use ESTs in maker\_gff: 1 = yes, 0 = no

altest\_pass=0 #use alternate organism ESTs in maker\_gff: 1 = yes, 0 = no

protein\_pass=0 #use protein alignments in maker\_gff: 1 = yes, 0 = no

rm\_pass=0 #use repeats in maker\_gff: 1 = yes, 0 = no

model\_pass=0 #use gene models in maker\_gff: 1 = yes, 0 = no

pred\_pass=0 #use ab-initio predictions in maker\_gff: 1 = yes, 0 = no

other\_pass=0 #passthrough anyything else in maker\_gff: 1 = yes, 0 = no

#-----EST Evidence (for best results provide a file for at least one)

est=est.fa #set of ESTs or assembled mRNA-seq in fasta format

altest=est.fa #EST/cDNA sequence file in fasta format from an alternate organism

est\_gff= #aligned ESTs or mRNA-seq from an external GFF3 file

altest\_gff= #aligned ESTs from a closly relate species in GFF3 format

#-----Protein Homology Evidence (for best results provide a file for at least one)

protein=prot.fa#protein sequence file in fasta format (i.e. from mutiple oransisms)

protein\_gff= #aligned protein homology evidence from an external GFF3 file

#-----Repeat Masking (leave values blank to skip repeat masking)

model\_org=all #select a model organism for RepBase masking in RepeatMasker

rmlib= Sru.lib#provide an organism specific repeat library in fasta format for RepeatMasker

repeat\_protein=/maker/data/te\_proteins.fasta #provide a fasta file of transposable element proteins for RepeatRunner

rm\_gff= #pre-identified repeat elements from an external GFF3 file

prok\_rm=0 #forces MAKER to repeatmask prokaryotes (no reason to change this), 1 = yes, 0 = no

softmask=1 #use soft-masking rather than hard-masking in BLAST (i.e. seg and dust filtering)

#-----Gene Prediction

snaphmm=Sru.hmm#SNAP HMM file

gmhmm= #GeneMark HMM file

augustus\_species=Srubrivinctus #Augustus gene prediction species model

fgenesh\_par\_file= #FGENESH parameter file

pred\_gff= #ab-initio predictions from an external GFF3 file

model\_gff= #annotated gene models from an external GFF3 file (annotation pass-through)

est2genome=1 #infer gene predictions directly from ESTs, 1 = yes, 0 = no

protein2genome=1 #infer predictions from protein homology, 1 = yes, 0 = no

unmask=0 #also run ab-initio prediction programs on unmasked sequence, 1 = yes, 0 = no

#-----Other Annotation Feature Types (features MAKER doesn't recognize)

other\_gff= #extra features to pass-through to final MAKER generated GFF3 file

#-----External Application Behavior Options

alt\_peptide=C #amino acid used to replace non-standard amino acids in BLAST databases

cpus=1 #max number of cpus to use in BLAST and RepeatMasker (not for MPI, leave 1 when using MPI)

#-----MAKER Behavior Options

max\_dna\_len=100000 #length for dividing up contigs into chunks (increases/decreases memory usage)

min\_contig=1 #skip genome contigs below this length (under 10kb are often useless)

pred\_flank=200 #flank for extending evidence clusters sent to gene predictors

pred\_stats=1 #report AED and QI statistics for all predictions as well as models

AED\_threshold=1 #Maximum Annotation Edit Distance allowed (bound by 0 and 1)

min\_protein=0 #require at least this many amino acids in predicted proteins

alt\_splice=0 #Take extra steps to try and find alternative splicing, 1 = yes, 0 = no

always\_complete=0 #extra steps to force start and stop codons, 1 = yes, 0 = no

map\_forward=0 #map names and attributes forward from old GFF3 genes, 1 = yes, 0 = no

keep\_preds=0 #Concordance threshold to add unsupported gene prediction (bound by 0 and 1)

split\_hit=10000 #length for the splitting of hits (expected max intron size for evidence alignments)

single\_exon=0 #consider single exon EST evidence when generating annotations, 1 = yes, 0 = no

single\_length=250 #min length required for single exon ESTs if 'single\_exon is enabled'

correct\_est\_fusion=0 #limits use of ESTs in annotation to avoid fusion genes

tries=2 #number of times to try a contig if there is a failure for some reason

clean\_try=0 #remove all data from previous run before retrying, 1 = yes, 0 = no

clean\_up=0 #removes theVoid directory with individual analysis files, 1 = yes, 0 = no

TMP= #specify a directory other than the system default temporary directory for temporary files

**Note:** The maker\_bopts.ctl (below) can be further edited to adjust Maker further to adjust stringency of the blast searches. For more partial genomes, the percent coverage threshold (protein or est) should be reduced.

### maker\_bopts.ctl

#-----BLAST and Exonerate Statistics Thresholds

blast\_type=ncbi+ #set to 'ncbi+', 'ncbi' or 'wublast'

pcov\_blastn=0.6 #Blastn Percent Coverage Threshold EST-Genome Alignments

pid\_blastn=0.6 #Blastn Percent Identity Threshold EST-Genome Aligments

eval\_blastn=1e-6 #Blastn eval cutoff

bit\_blastn=40 #Blastn bit cutoff

depth\_blastn=0 #Blastn depth cutoff (0 to disable cutoff)

pcov\_blastx=0.6 #Blastx Percent Coverage Threshold Protein-Genome Alignments

pid\_blastx=0.6 #Blastx Percent Identity Threshold Protein-Genome Aligments

eval\_blastx=1e-6 #Blastx eval cutoff

bit\_blastx=30 #Blastx bit cutoff

depth\_blastx=0 #Blastx depth cutoff (0 to disable cutoff)

pcov\_tblastx=0.6 #tBlastx Percent Coverage Threshold alt-EST-Genome Alignments

pid\_tblastx=0.6 #tBlastx Percent Identity Threshold alt-EST-Genome Aligments

eval\_tblastx=1e-6 #tBlastx eval cutoff

bit\_tblastx=40 #tBlastx bit cutoff

depth\_tblastx=0 #tBlastx depth cutoff (0 to disable cutoff)

pcov\_rm\_blastx=0.6 #Blastx Percent Coverage Threshold For Transposable Element Masking

pid\_rm\_blastx=0.6 #Blastx Percent Identity Threshold For Transposbale Element Masking

eval\_rm\_blastx=1e-6 #Blastx eval cutoff for transposable element masking

bit\_rm\_blastx=30 #Blastx bit cutoff for transposable element masking

ep\_score\_limit=20 #Exonerate protein percent of maximal score threshold

en\_score\_limit=20 #Exonerate nucleotide percent of maximal score threshold

The maker\_exe.ctl simply tells Maker where to find the executables and should not need to be edited.

Once maker\_opts.ctl and maker\_bopts.ctl have been viewed, run Maker.

1. Run maker by editing then launching the following Qsub script
   1. $qsub maker.qsub

#!/bin/bash

#PBS -k o

#PBS -l nodes=1:ppn=8,walltime=160:00

#PBS -N MAKER\_output

#PBS -j oe

module load mvapich2

module load Maker2.28

workdir=/home/YOUR\_USER\_Name/maker

cd $workdir

mpirun -n 8 maker

# Maker's Results

Maker will take each scaffold and place it in its own directory along with the output that Maker generates for that scaffold. The three most important files that are created are the gff, Maker's predicted transcripts, and Maker's predicted proteins.

Maker's directory structure:

Maker divides the scaffold directories into several layers of subdirectories so the filesystem will not be slowed down by having to handle hundreds or even thousands of directories at one time.

The most efficient way to make use of Maker's output is by combining Maker's output into a relatively small number of large files. We will typically want to upload the genome to CoGo and then a single file containing the annotations (highlighted option below). If the file is very large, psftp may be necessary to use for the transfer.

1) Gather all of the GFFs into one conglomerate GFF.

In the testScaf500.maker.output directory

Several options:

Everything (including Maker's prediction, all evidence, masked repeats, and DNA)

$module load Maker2.28

$gff3\_merge -d testScaf500\_master\_datastore\_index.log

Everything, but no DNA at the end

$module load Maker2.28

$gff3\_merge -n -d testScaf500\_master\_datastore\_index.log

You should see a new file called testScaf500.all.gff.

Only Maker annotations

$module load Maker2.28

$gff3\_merge -n -g -d testScaf500\_master\_datastore\_index.log

# Copy the gff file from unix to your desktop

For mac go to your mac terminal and use scp to copy to your desktop.

$scp testScaf500.all.gff [username@192.112.102.21:/path](mailto:username@192.112.102.21:/path)

For windows, double-click on psFTP on your computer and connect to the Juniata-HHMI cluster:

psftp>open username@192.112.102.21

Notice that your working directory is shown. Next transfer the file using the following command. The file will get transferred to the same folder where psFTP was located.

psftp>get path/testScaf500.all.gff

### GFF file format

General Feature Format (GFF) is a standard file format, meaning that it is supported and used by several programs and appears in the same form regardless of what program generated it. GFF is a tab delimited file featuring nine columns of sequence data.

|  |  |  |
| --- | --- | --- |
| Column | Feature | Description |
| 1 | Seqname | The name of the sequence |
| 2 | Source | The program that created the feature |
| 3 | feature | What the sequence is (gene, contig, exon, match) |
| 4 | Start | First base in the sequence |
| 5 | End | Last base in the sequence |
| 6 | Score | Score of the feature |
| 7 | Strand | + or - |
| 8 | Frame | 0,1, or 2 first, second, or thrid frame |
| 9 | Attribute | List of miscellaneous features not covered in the first 8 columns, each feature separated by a ; |

Example gff (Shows part of a single exon maker gene found on scaffold 1)



**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

One can also do analysis on the following files with various programs.

1) Gather all the predicted proteins (the predicted proteome).

In the testScaf500\_datastore directory

$cat ./\*/\*/\*/\*.proteins.fasta > genome.predictedproteins.fasta

2) Gather all the predicted transcripts (the predicted transcriptome)

In the same directory

$cat ./\*/\*/\*/\*.transcripts.fasta > genome.predictedtranscripts.fasta

# III. Using Fathom to assess Annotation Quality

This tutorial uses the output from Maker after genome.fa was annotated. Also, running any of the listed commands without any additional input (or running them with the -h option) will display the help message including a description of the program, the usage, and all options.

1) Gather all of the GFFs into one inclusive file.

In the genome.maker.output directory load Maker, grab info from the Maker run, and see what the options you just used meant.

$module load Maker2.28

$gff3\_merge -g -d genome\_master\_datastore\_index.log

$gff3\_merge –help

2) Convert the gff to zff (a proprietary format used by fathom)

$maker2zff -n genome.all.gff

This will create two new files genome.ann & genome.dna

The .ann file is a tab delimited file similar to a gff with sequence information. The .dna file is a fasta file that corresponds to the sequences in the .ann file.

3) First, determine the total number of genes and total number of errors there were. The results will appear on your screen. Enter them in the table below.

$module load SNAP

$fathom –validate genome.ann genome.dna

4) Next, calculate the number of Multi and Single exon genes as well as average exon and intron lengths. Enter them in the table below.

$fathom –gene-stats genome.ann genome.dna

5) Categorize splits the genes in the genome.ann and genome.dna files into 5 different categories: Unique (uni.ann uni.dna), Warning (wrn.ann uni.dna), Alternative splicing (alt.ann alt.dna), Overlapping (olp.ann olp.dna), and Errors (err.ann err.dna).

$fathom –categorize 1000 genome.ann genome.dna

Explanation of categories

|  |  |
| --- | --- |
| Category | Explanation |
| Unique | Single complete gene with no problems:  Good start and stop codons, canonical intron exon boundaries (if a multi-exon gene) no other genes overlapping it or any other potential problems |
| Warning | Something about the gene is incorrect: missing start or stop codon, does not have a canonical intron exon boundary, or has a short intron or exon (or is a short gene if it has a single exon) |
| Alternative splicing | Genes that are complete, but have alternative splicing |
| Overlapping | Two or more genes that overlap each other |
| Error | Something occurred and the gene is incorrect. Usually caused by miss-ordered exons. |

6) To complete the table below, examine the file produced from step (5) that corresponds to each tabled category.

### Using Fathom, complete the following table using your data

|  |  |  |
| --- | --- | --- |
|  | Count | Percent of Total |
| Total |  |  |
| Complete Genes |  |  |
| Warnings |  |  |
| Alternative splice |  |  |
| Overlapping |  |  |
| Errors |  |  |
| Single Exon |  |  |

# If there’s time: Training SNAP & Augustus

This tutorial assumes you are using a multifasta file named genome.fa you could, however, use any fasta file simple replacing genome.fa with the name of the fasta file being used. This also assumes that you are using the custom repeat library created in the RepeatScout tutorial and a protein file named proteins.fa and ests names est.fa. We will use programs associated with MAKER to obtain the high quality gene annotation file to build our HMM files.

To perform the protein2genome and est2genome annotations, run maker as described in above, but either turn off SNAP and Augustus options by leaving those blank, or substitute a file from a closely related species if available.

To construct the SNAP HMM file do the following steps.

1) Once maker has finished, the results need to be gathered.

$module load Maker2.28

$cd genome.maker.output

$gff3\_merge -d genome\_master\_datastore\_index.log -g

2) Make a new directory for training SNAP in and put the gff in there.

$mkdir SNAPTraining

$mv genome.all.gff SNAPTraining

$cd SNAPTraining

3) Convert the gff to zff (the file format that snap uses for training).

$maker2zff -n genome.all.gff

This will create two new files: genome.ann and genome.dna

4) fathom and forge will be used to train snap. Fathom has several subroutines that help filter and analyze gene annotations. First, examine the annotations to get an impression of what annotations there are.

$fathom -validate genome.ann genome.dna

$fathom -gene-stats genome.ann genome.dna

5) Next, split the annotations into four categories: unique genes, warnings, alternative spliced genes, overlapping genes, and errors.

$fathom -categorize 1000 genome.ann genome.dna

6) Next, export the genes.

$fathom -export 1000 uni.ann uni.dna

This will generate four new files (export.aa, export.ann, export.dna, and export.tx)

7) Make a new directory for the new parameters.

$mkdir parameters

$cd parameters

8) Generate the new parameters. The command below will run forge on the directory above parameters, but put the output in the working directory (parameters).

$forge ../export.ann ../export.dna

9) Generate the new HMM (the file that snap uses to generate gene predictions), by moving into the directory above parameters, and entering the command that follows.

$cd ..

$hmm-assembler.py genome parameters > genome.hmm

# Training Augustus

1) Go to <http://bioinf.uni-greifswald.de/webaugustus/training/create>

2) In the designated spaces fill in your email and the species name

3) Upload export.dna as the genome file.

4) Upload export.aa as the protein file.

5) Type in the verification string and click “Start Training”

6) After the Augustus web training finishes, download the directory of results.

If using HHMI Cluster: Rename the directory by the species name and copy into /share/apps/Installs/Sickler/maker/exe/augustus/config/species directory.

You will need to use that species name in your maker\_opts.ctl file.

Assessment/Discussion

Are all your genes complete? Is this a good annotation? How could the annotation be improved?

Time line of module

Three hour lab

References

### Maker

Cantarel BL, Korf I, Robb SM, Parra G, Ross E, Moore B, Holt C, Sánchez Alvarado A, Yandell M. 2008.MAKER: an easy-to-use annotation pipeline designed for emerging model organism genomes. Genome Res.18:188–196.

Holt C, Yandell M. MAKER2: an annotation pipeline and genome-database management tool for second-generation genome projects. BMC Bioinformatics. 2011;12:491. doi: 10.1186/1471-2105-12-491.

### SNAP

Korf I. 2004. Gene finding in novel genomes. BMC Bioinformatics5:59

### Augustus

Stanke M, Waack S. Gene prediction with a hidden Markov model and a new intron submodel.Bioinformatics. 2003;19:ii215–ii225. doi: 10.1093/bioinformatics/btg1080.

Stanke M, Schöffmann O, Morgenstern B, Waack S. Gene prediction in eukaryotes with a generalized hidden Markov model that uses hints from external sources. BMC Bioinformatics. 2006;7:62. doi: 10.1186/1471-2105-7-62.

Module 6. Whole Genome Annotation: Misc

Background

Grep can be used to grab information from one or many files at once.

Command line Blast allows one to search the genome multifasta file for sequence motifs.

Goals

* Use Grep to pull sequences out of a large multifasta file
* Run blast from the command line

V&C core competencies addressed

1) Ability to apply the process of science: Evaluation of experimental evidence, Developing problem-solving strategies

2) Ability to use quantitative reasoning: Applying statistical methods to diverse data, Managing and analyzing large data sets

3) Applying informatics tools, Managing and analyzing large data sets

GCAT-SEEK sequencing requirements

None

Computer/program requirements for data analysis

Linux OS, Fathom, BlastIf starting from Window OS: Putty

If starting from Mac OS: SSH

Protocols

# I. Grep Tutorial

Grep finds and prints every line that contains the pattern being searched for in one or more files.

Usage: grep [options] “pattern” filename

|  |  |  |
| --- | --- | --- |
| Command | Function | Example Usage |
| Grep | Searches for lines in the file containing the search pattern | grep “GENE” annotation.gff |
| grep –i | Same as grep but is not case sensitive | grep -i “$think of something” $\_ |
| grep –c | Counts the number of times the pattern appears | grep -c “>” proteins.fasta |
| grep -An | Prints the line matching the pattern and n lines after the pattern | grep -A1 “scaffold1” genome.fasta |

1. First , make a new folder from your home directory, move into it, and then copy the Maker control and other sample files from our shared directory into your directory.

$mkdir 6\_grepBlast

$cd 6\_grepBlast

$cp /share/apps/sharedData/GCAT/6\_grepBlast/\* .

1. The file apples.txt contains:

APPLE

pineapple

apples

Run:

$grep “APPLE” apples.txt

grep finds “APPLE” and will print:

APPLE

1. Using “grep –i” makes a search not case sensitive. Predict what the following will return, and then run the command:

$grep -i “APPLE” apples.txt

4. The option “$grep –c” will count the number of lines the pattern appears in a file. This is very useful for counting the number of sequences in a multifasta file. Open up (using nano) the fasta file below, named “test1.fa”

>scaffold\_0

GCATGCATGCATGCATGCATGCATGCATGCATGCATGCATGCATGCATGCATGCATGCATGCATGCATGCATGCATGCATGCATGCATGCATGCATGCATGCATGCATGCATGCATGCATGCATGCATGCATGCATGCATGCATGCATGCATGCATGCATGCATGCATGCATGCATGCATGCATGCATGCATGCATGCATGCATGCATGCATGCATGCATGCATGCATG

>scaffold\_1

AGCCCTCCAGGACAGGCTGCATCAGAAGAGGCCATCAAGCAGGTCTGTTCCAAGGGCCTTTGCGTCAGGTGGGCTCAGGATTCCAGGGTGGCTGGACCCCAGGCCCCAGCTCTGCAGCAGGGAGGACGTGGCTGGGCTCGTGAAGCATGTGGGGGTGAGCCCAGGGGCCCCAAGGCAGGGCACCTGGCCTTCAGCCTGCTCAGCCCTGCCTGTCTCCCAGATCACTGTCCTTCTGCCATGGCCCTGTGGATGCGCCTCCTGCCCCTGCTGCGCTGCTG

Close the file and run,

$grep -c “>” test1.fa

Does your answer make sense?

5. The “-A n” option will print the line matching the pattern and n number of lines after it. So, running the command:

$grep –A 1 “scaffold\_0” test1.fa

…will print out the name and sequence of a scaffold to the screen. To capture the scaffold to a new file called scaffold\_0.fa, use “>” at the end of the command.

$grep –A 1 “scaffold\_0” test1.fa > scaffold\_0.fa

6. You can use grep to see how many maker genes were predicted in your annotation. Move into your maker.output folder, and hit $ls to see the name of your files. Use nano to open the file ending with “all.gff,” and look at its structure. Find the word “gene” in the third and last columns. Notice that the word “gene” is found only once in the third column per gene, and is always proceeded by blank space. We can count the number of genes using:

$grep –c –E “\sgene” testScaf500.all.gff

Recall that “-c” counts lines with matching criteria. “-E” allows for regular expressions in the search term. Using regular expressions we can give general meanings to characters. For example using “\” before “s” gives “s” the special meaning, of whitespace. We searched for the word “gene” preceded by whitespace. Try the search without “-c” to look at the lines grep found. Notice that these lines tell you where the genes are in summary.

**Some notes on grep:**

If a line contains the pattern more than one time, the line will be printed or counted only once.

### Other options:

|  |  |
| --- | --- |
| -n | Print the line number for lines that match the pattern |
| -B n | Print lines before the pattern (opposite of -An) |
| -H | Print the file's name before each line (when searching multiple files) |
| -f file | Read the pattern from a file line by line |
| -v | Will print lines that don't match the pattern |
| -x | Print the exact match. The line can only |
| -m n | Stop after finding n matches |
| egrep | Uses the full set of regular expressions |
| fgrep | Does not use regular expressions and only searches of literal strings. (Faster than grep) |

**Regular expressions:**

|  |  |
| --- | --- |
| \* | Wildcard, any character |
| . | Any character other than new line (\n) |
| \t | Tab |
| \s | Whitespace (space, Tab, new line) |
| ^ | Beginning of line |
| $ | End of line |
| \n | Carriage return |

More regular expressions can be found at:

<http://regexlib.com/CheatSheet.aspx?AspxAutoDetectCookieSupport=1>

# II. Command line Blast Tutorial

The Basic Local Alignment Search Tool or BLAST is easily the most common and widely used piece of bioinformatical software. BLAST can perform several different types of alignments with several different forms of output. BLAST can use either publicly available, previously created databases to compare the sequence of interest too, or it can be used with a custom database.

### Creating a custom BLAST database

1) We will use a genome file to make a custom reference database that we can then search against with a protein query (blastn).

$module avail

$module load BLAST2.28

2) Stay in your /grepBlast folder and create a BLAST database from a reference set of fasta files using test2.fa.

$makeblastdb -in [reference.fa] -dbtype [nucl or prot] -title [reference] -out [reference]

e.g.

$makebastdb –in test2.fa –dbtype nucl -title reference –out reference

This will create three database files all starting with “reference” and ending in certain file extensions. If the initial sequences were proteins, dbtype would be set to prot and if they were nucleotides, dbtype would be set to nucl.

### Use BLAST to align the sequence of interest to the custom database

1) Run Blastn as follows:

$[t]blast[n p x] -query test1.fa -db reference -out blastresults.txt -outfmt 0

**e.g.**

$blastn -query test1.fa -db reference -out blastresults.txt -outfmt 0

The outfmt (outformat) option tells BLAST what format to output the data in. Pairwise (default or –outfmt 0) is a good way to visualize results. 7 (-outfmt 7) is a tabular output with comments indicating column titles. This is a good format for exporting to a spreadsheet.

BLAST also has options to control stringency of match between subject and query. To see the options call up the help documentation for the type of blast you are running. For example:

$blastn –help

For example, to run the same query, but change the e-value to 1 x 10E-10:

$blastn -query test1.fa -db reference –evalue 1e-10 -out blastresults.txt -outfmt 0

### Blast types

|  |  |
| --- | --- |
| Blastn | Nucleotide to nucleotide aligment |
| Blastp | Protein to protein aligment |
| Blastx | Translated nucleotide subject to protein database alignment. |
| Tblastx | Translated nucleotide to translated nucleotide database alignment. |
| Tblastn | Protein to translated nucleotide alignment. |

Assessment/Discussion

What did you think of grep?

References

Korf I. 2004. Gene finding in novel genomes. BMC Bioinformatics5:59 [SNAP and FATHOM]

Haddock SHD, Dunn CW. 2011. Practical computing for biologists. Sinauer Associates

Korf I. Unix and Perl Primer for Biologists. korflab.ucdavis.edu/Unix\_and\_Perl/unix\_and\_perl\_v3.0.pdf‎

Module 7. Comparative Genomics using CoGe

Background

Genomes are understood by comparison to other genomes.

Goals

* Finding syntenic regions among closely related genomes, focused on a target gene
* Examine regions up close across multiple genomes using “MicroSynteny” tools
* Draw phylogenetic trees using focal gene regions
* Examine evidence for positive selection in a pairwise analysis between species

V&C core competencies addressed

1) Ability to apply the process of science: Evaluation of experimental evidence, Developing problem-solving strategies

2) Ability to use quantitative reasoning: Applying statistical methods to diverse data, Managing and analyzing large data sets

3) Applying informatics tools, Managing and analyzing large data sets

GCAT-SEEK sequencing requirements

None

Computer/program requirements for data analysis

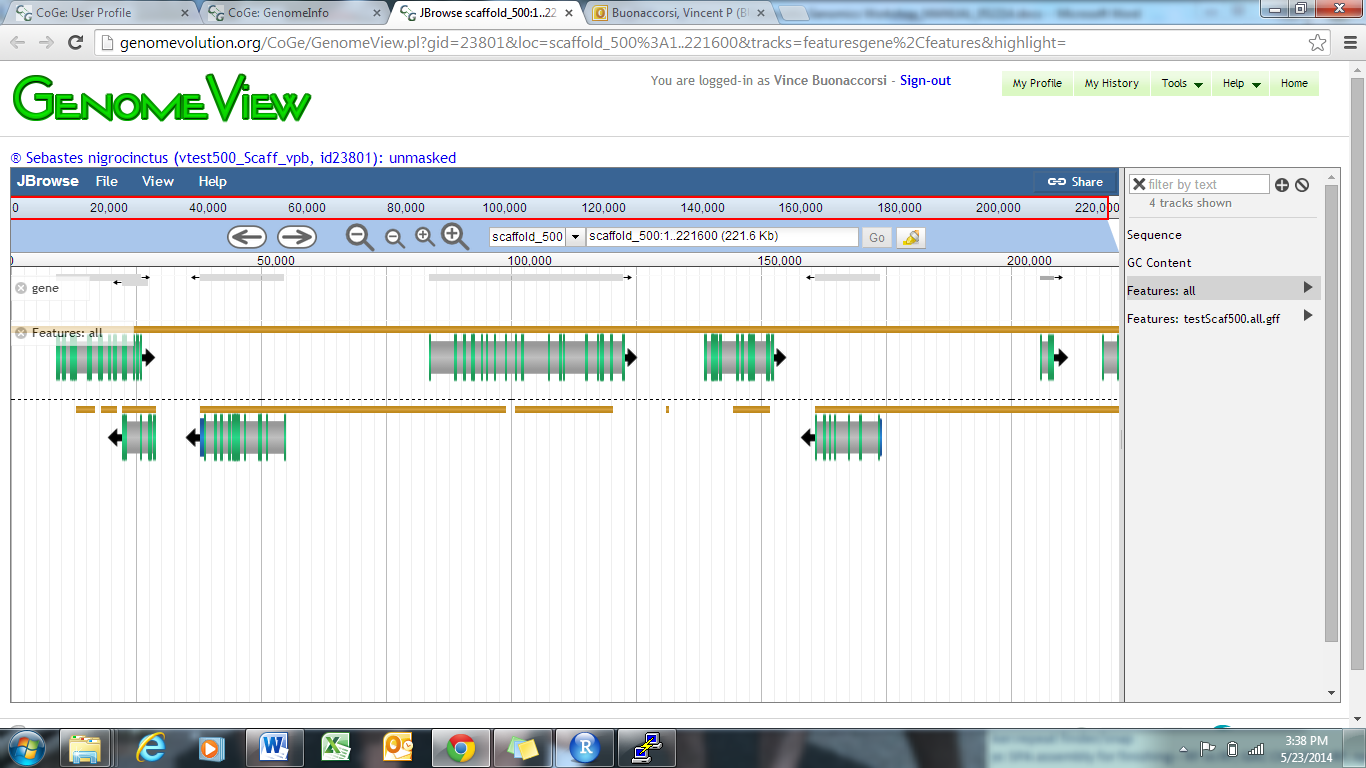
Linux OS, Fathom, BlastIf starting from Window OS: Putty

If starting from Mac OS: SSH

Protocols

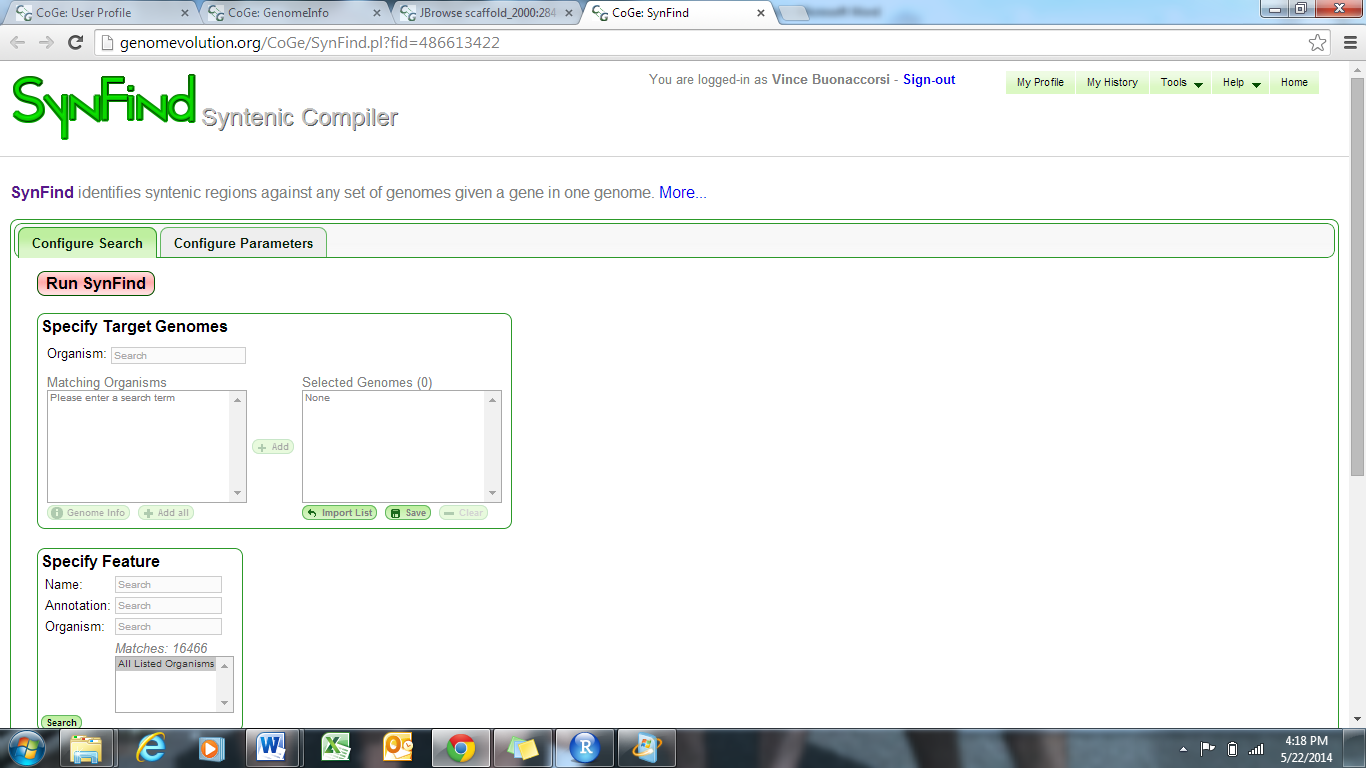
## Finding syntenic regions among closely related genomes, focused on a target gene

1. If you haven’t already done so, register for an iplant account at <http://user.iplantcollaborative.org/>
2. Use a Firefox or Chrome internet browser to get to: www.genomevolution.org
3. Sign in to CoGe using your iplant login information
4. Select the “My profile” button in the top right of the CoGe home screen.
5. Select the “Create” button at the lower left of your screen, and chose, “load genome”
6. When the new window appears, enter the scientific name “Sebastes nigro...” and select the completed name as it appears below.
7. In the “version” box, enter “testScaf500\_” followed by your initials. (e.g. testScaff500\_vpb)
8. In the source box, enter your initials again.
9. Select the green “add data” button below these boxes and navigate (Upload/choose file) to your folder for this lab to upload the S. nigrocintus testScaf500.fa file. Hit load genome after the data has been transferred to the iplant server. Hit continue to genome info when it finishes.
10. At the lower left of the next (Genome information) screen, select the green “Load Gene Annotation” button and upload the correct gff3 file from your working folder. Be careful to match the gff file with the fasta file that was used to create it.
11. Select the “View” button to view the “genome” individually from the genome information page.

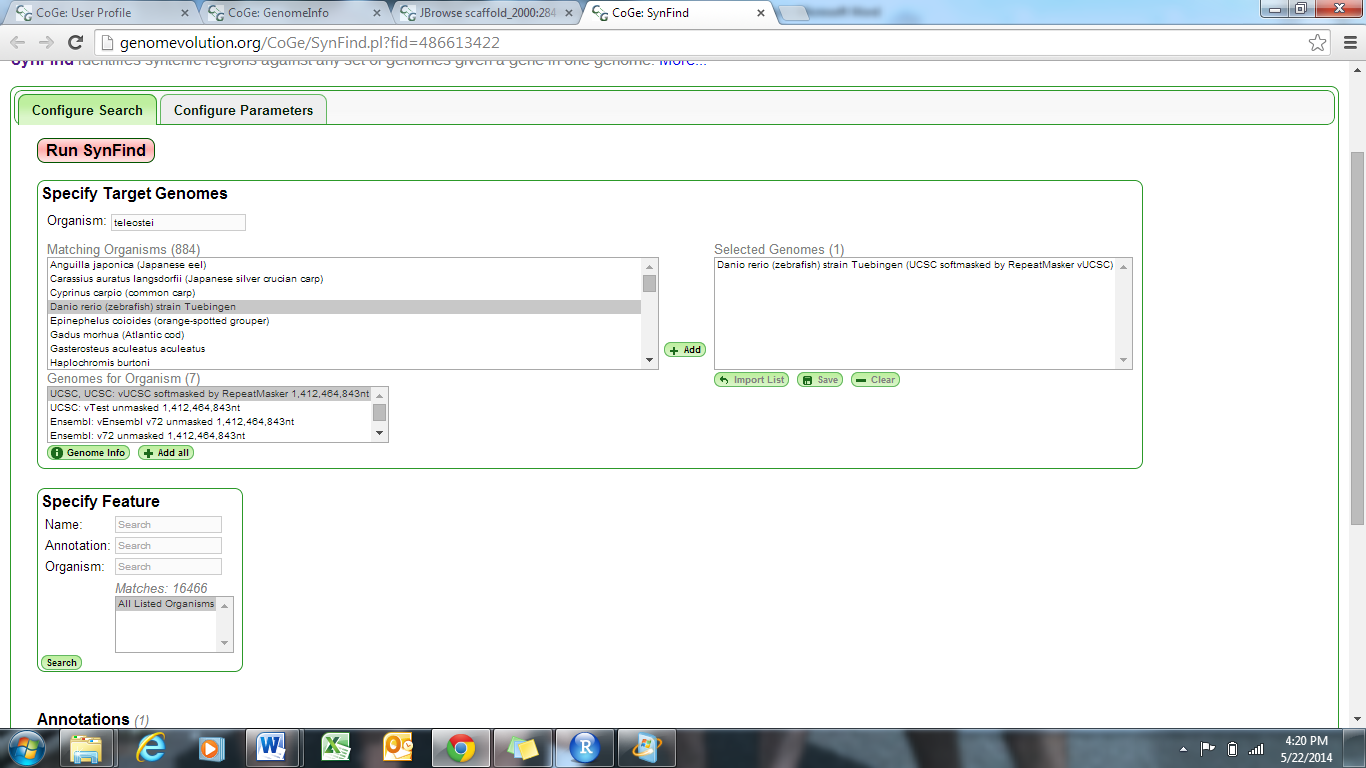


1. Find introns, exons, & UTRs. Note the Features box on the right needs to be selected to see gene models.
2. Click on the big gene in the middle. When the window below appears, select Fasta to view the coding sequence, close the window, and then select SynFind





From this window, enter Teleostei in the “Specify Target Genomes” box to pull up closely related fishes to our focal species.

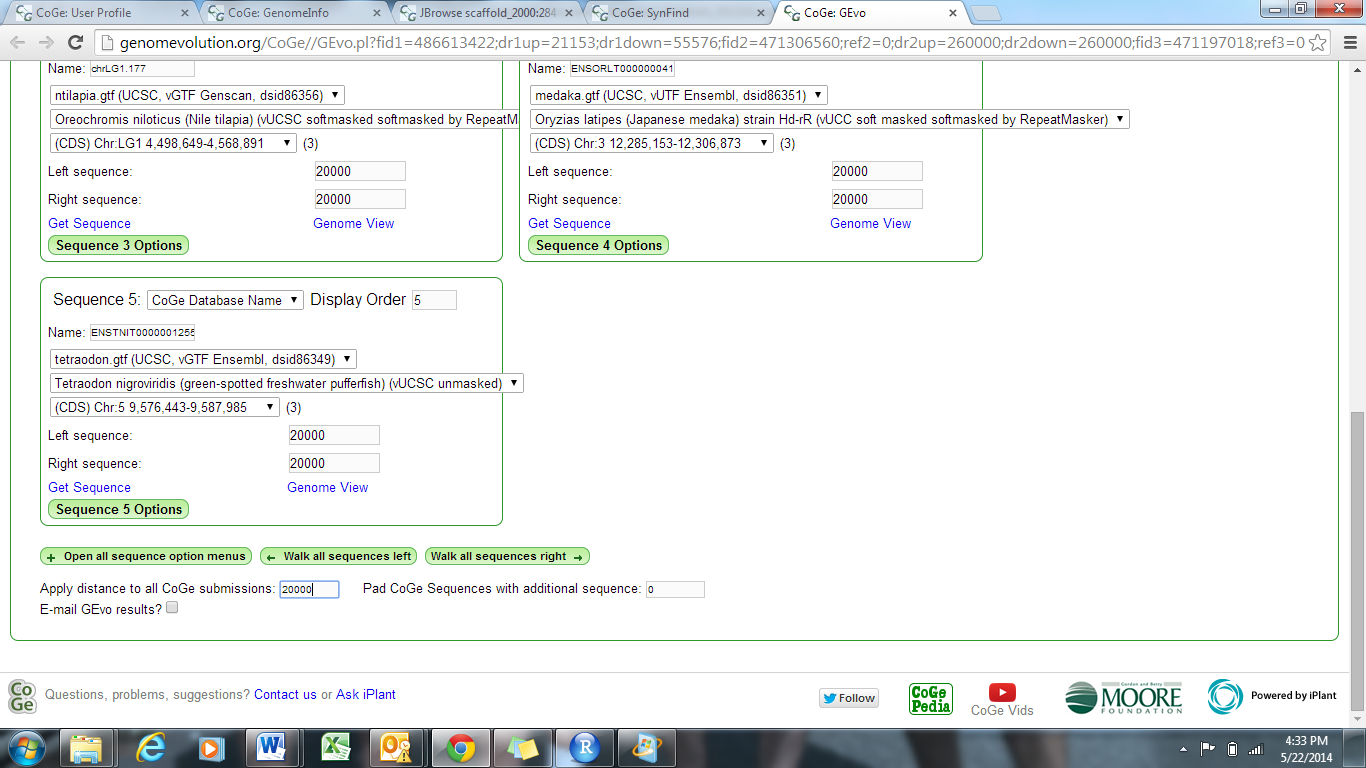


From this window, slowly scroll down on the left until the “+Add” button becomes bold (annotations are available). Hit Add to upload this species and all other non-mitochondrial options that become bolded when scrolled over. It will be about a dozen species. Then go to the configure parameters to examine what kind of search is being done. Note that the minimum number of gene hits for a syntenic block is set to 4 by default, return to the other tab, and hit Run SynFind. Notice that a large gene window is searched for, of 40 genes. This kind of analysis is most informative with medium to high quality genome assemblies.



Syntenic regions in four of the 12 species were found.

* Select “Generate Fasta Sequences”. Notice that CoGe extracts the coding sequences from all genomes. Selecting the “Protein” link will translate the sequences. DNA or protein sequences can be easily downloaded as multifasta text files for further analysis.
* Flip back to the SynFind browser tab. Select the “Compare and visualize in GEvo” link to look at the comparisons in genomic context using a Micro-synteny tool.



## Examine regions up close using “MicroSynteny” view.

The next page takes a little time to load. Go ahead, scroll down and change the “distance” analyzed to the left and right of each focal gene to 20000, then scroll up and hit the “run GEvo analysis button”. When the next page shows up, the focal gene’s exons are shown in yellow. Neighboring gene exons are shown in green.



Boxes above your gene show blocks of microsynteny. Select a red box towards the middle of the screen, then hold shift and select another. For mac users, use COMMAND-SHIFT while clicking to select all boxes. Scroll down and see the similarity to that species. Hit “Clear connectors” to remove the transluscent boxes showing matching sequences, and select bars for the top species. Double click on a box to pull up a comparison of the regions, and select “full annotation” to see a detailed alignment. Hit “clear connectors” to undo selections.



Now scroll down, hit “open all sequence option menus”, and Mask Non-CDS regions, like above in each species. Go to the algorithm tab and select tblastx as the search tool and rerun CoGe.



Play with the CoGe analysis options and try to find missing exons in one species or another.

## Draw phylogenetic trees for focal gene regions

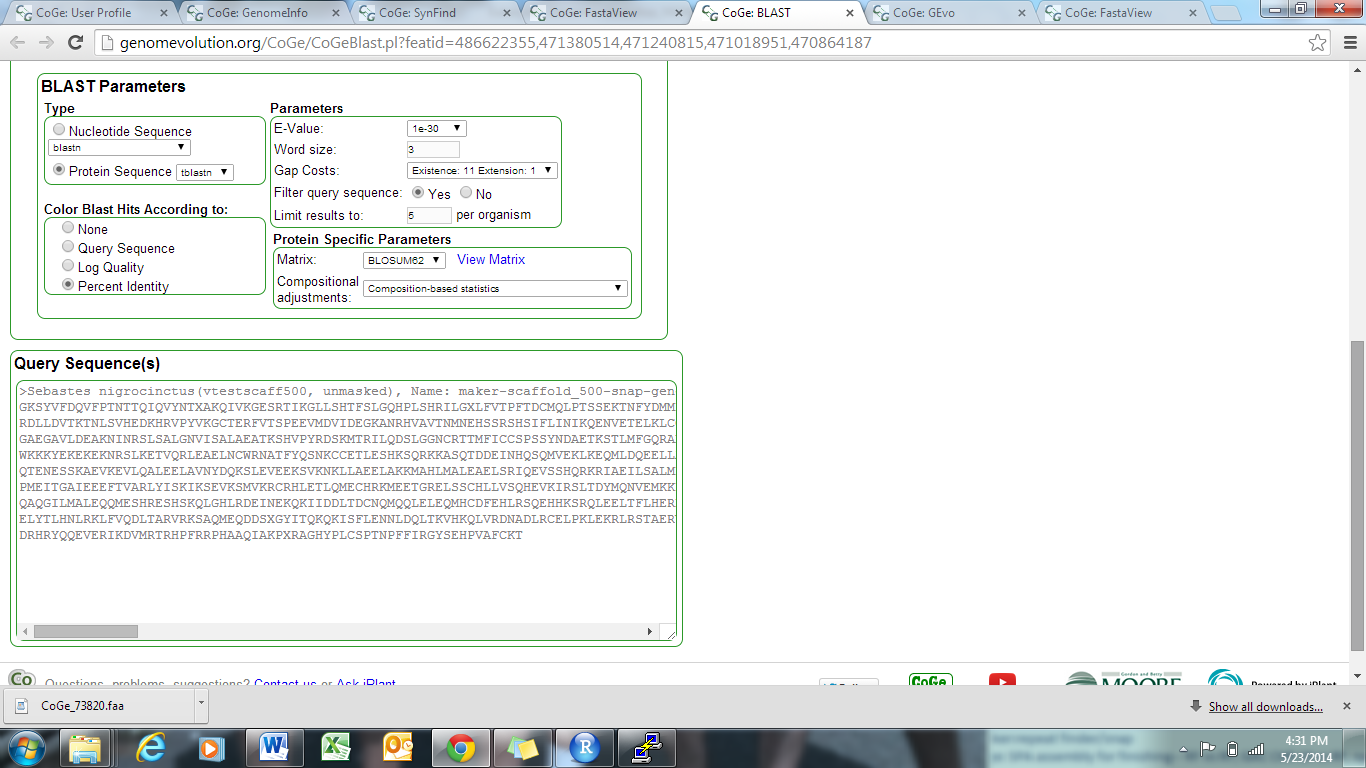
Go back to the SynFind page and select Generate Fasta Sequences.



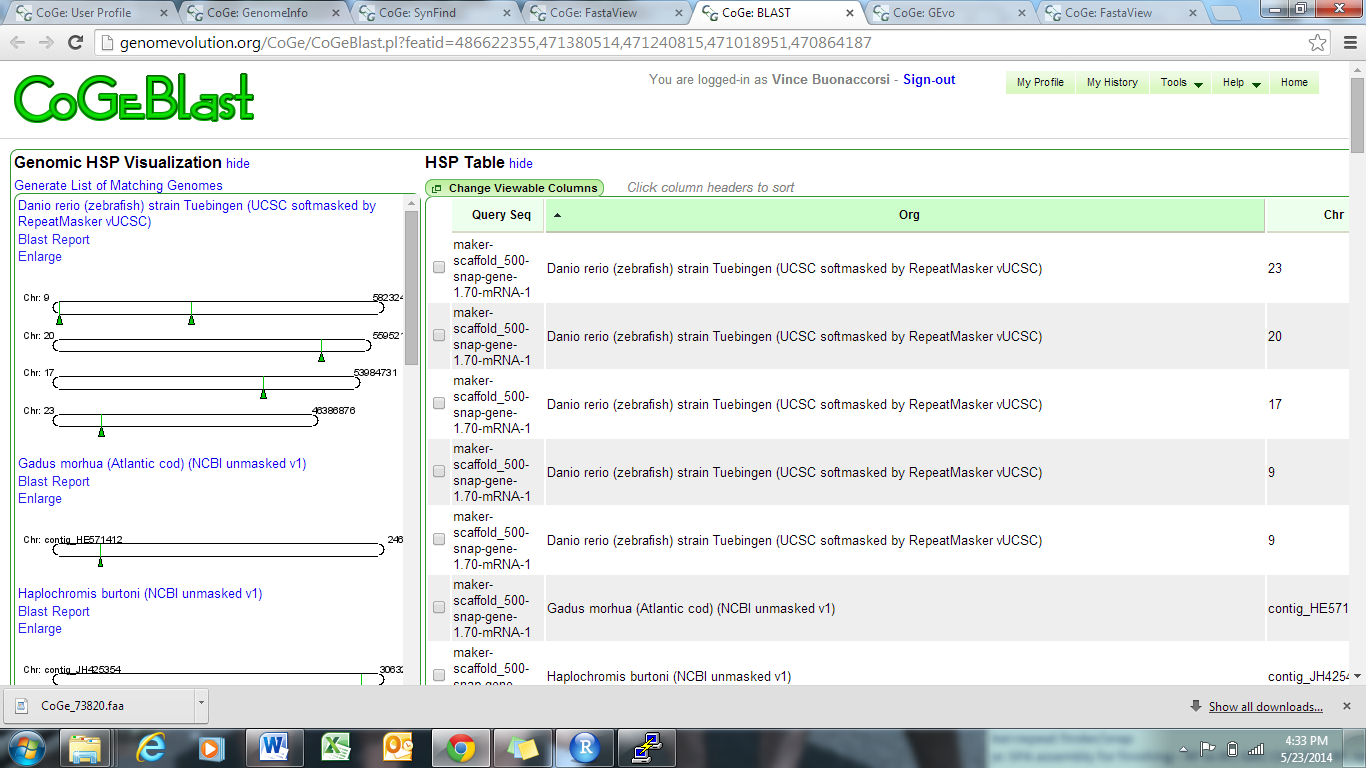
In the FastaView page, select CoGe Blast.



Delete all seqs but the rockfish at the top. Translate the sequence by hitting “Protein Sequence”, then add fish genomes as above, and adjust parameters to your liking. Be sure to include the whole genome of the species that your query sequence is from if available. Make sure to Limit results to 3 per organism. Then hit Run CoGe blast.



The genomic location for hits is now shown.



You can look in detail at each hit in its genomic context.



Select all and “send selected to phylogenetics”. Scroll down on the FeatView window and select “phylogeny.fr”

## 

Assessment/Discussion

Examining your phylogenetic tree, is there evidence that the gene you selected belongs to a multigene family? Teleost genomes were duplicated about 50 million years ago. Do you see evidence for the duplication?

Module 8: STACKS

**Introduction**

Stacks is a computational pipeline designed to analyze all types of reduced representation sequencing data. The Stacks Pipeline can be used with or without the assistance of a reference genome. The pipeline begins with raw sequencing data and continues to population summary statistics, phylogenetic trees, and locus-specific haplotypes as well as various other results.

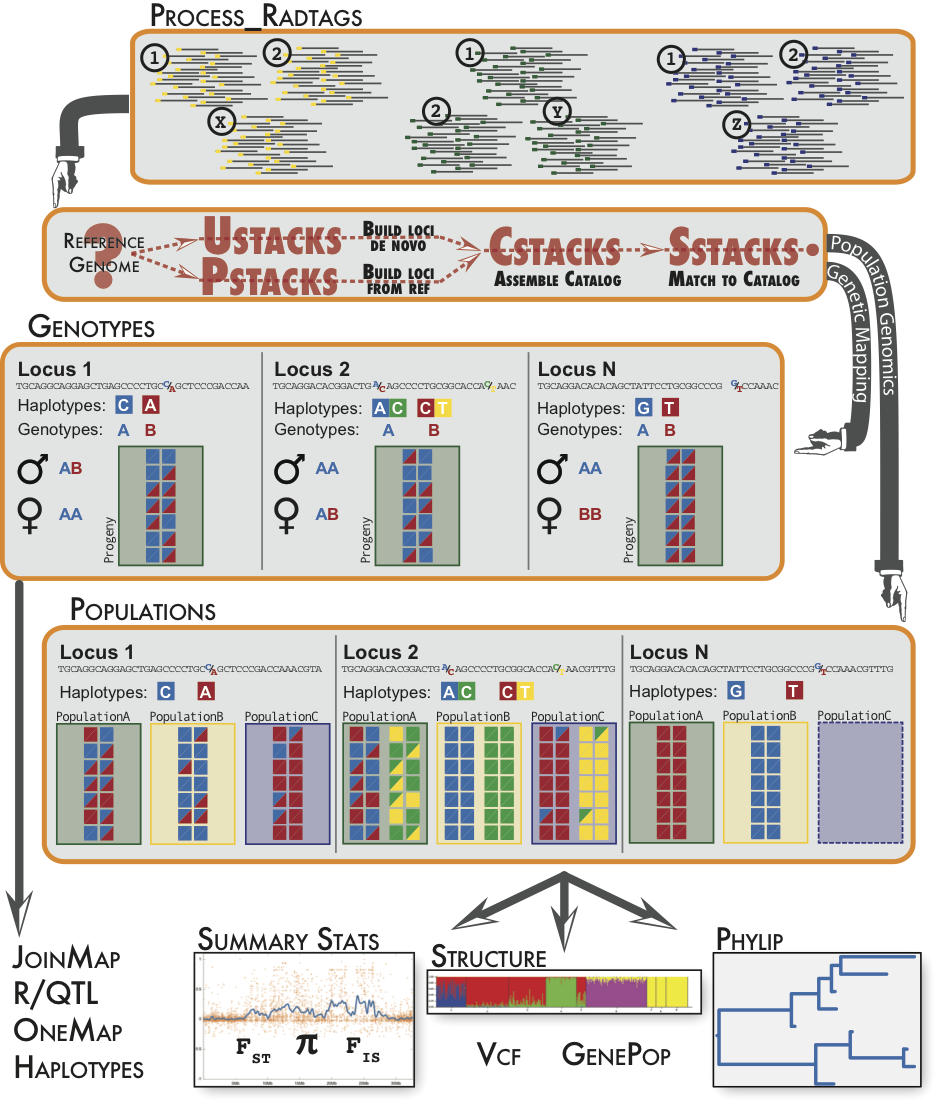
The files that we will be working with come from a genotyping by sequencing (GBS) project done on males and females from a species of Pacific rockfish, *Sebastes chrysomelas*. The aim of the project was to identify sex-linked markers.

Genomic DNA was digested using the Peterson et al. (2012) double digestion (ddRADseq) protocol using the enzymes sphI and mluCI by the Indiana University genome core facility. The digested DNA was then sequenced with 100bp paired-end reads with a target insert size of 250bp to 350bp. Sequenced reads were then processed and analyzed using the Stacks Pipeline.

Library Design Overview (https://docs.google.com/document/d/14LzChQSHRMghGaWL55EYsZ09Ve-3ncHIJzv6TLYvBcg/edit?hl=en\_US)



The Stacks Pipeline (<http://creskolab.uoregon.edu/stacks/manual/>)



**Running Stacks**

Note on managing directories

As you utilize the Stacks Pipeline, a large number of intermediate files will be created. Create new directories for each part of the pipeline to keep results organized. It is also important to give the directories short, informative names so you can easily manage them and understand what is in them. No spaces are allowed in file or directory names, so use “\_” or useCamelNotation to separate words.

Copy the files into your home directory

Make a new directory and move into it.

$ mkdir 8\_stacks

$ cd 8\_stacks

Copy the example files into your 8\_stacks directory.

$ cp -r /share/apps/sharedData/GCAT/8\_stacks/\* .

Demultiplexing

The first step of the Stacks Pipeline is to sort pools of raw reads into quality-filtered individuals (aka demultiplex). Stacks can demultiplex samples either using just a single sequencing barcode or by using a combination of barcode and index. ddRADseq protocols generally use both barcodes and indices in sequencing library formation to distinguish individuals. This reduces the need for longer, more expensive barcodes that would be needed if only using barcodes to distinguish individuals. In our case, the Indiana U genome core facility has already split out pools of sequences by index, and removed the index sequence from the reads.

Demultiplexing the samples

First, the samples must be unpacked and expanded.

$ tar -xvzf RadSeqSamples.tar.gz

There should now be a new folder RadSeqSamples with two files RadSeq\_1.fq and RadSeq\_2.fq.

Now, the samples can be demultiplexed. You need to make a folder first for the Demultiplexed files to go into.

$ mkdir Demultiplexed

Do not move into the new directory.

$ module load stacks1.04

$ process\_radtags -1 ./RadSeqSamples/RadSeqSamples.R1.fastq -2 ./RadSeqSamples/RadSeqSamples.R2.fastq -P -o Demultiplexed -q -r -b ./barcodes.txt --renz\_1 sphI --renz\_2 mluCI

-p specifies the path to a directory with raw reads in them, -P means the reads in the directory are paired, -o is the output directory, -q and -r filter reads for quality and attempt to restore cut-sites, barcodes, indices, and enzyme sites, --renz\_1 and --renz\_2 are the restriction enzymes used.

These commands will demultiplex the samples in RadSeqSamples, treat them as paired reads, discard low quality reads, attempt to restore cut sites, barcodes, and indices, and place the demultiplexed reads in the directory Demultiplexed.

Since we will be working with a set of files that were already demultiplexed, we will be doing the remainder of our work in the Samples directory.

Aligning reads to a reference genome using Bowtie 2

To make the Stacks Pipeline run faster and to aide in downstream analysis, the reads can be aligned to a reference genome. Bowtie 2 is a whole genome alignment tool designed to be fast and produce high quality results.

1) Construct the Bowtie 2 index files using the reference genome.

To save on time, this step has already been done.

$ module load Bowtie2-2.1.0

$ bowtie2-build -c RubriAll.fa RubriBowtie2

The -c option says that the sequence files are given on the command line instead of as a list which the program expects. This creates several files: RubriBowtie2.1.bt2, RubriBowtie2.2.bt2, RubriBowtie2.3.bt2, RubriBowtie2.4.bt2, RubriBowtie2.rev.1.bt2, and RubriBowtie2.rev.2.bt2

2) Run Bowtie 2

Copy the files to the Samples directory.

$ cp ./BowtieFiles/\* ./Samples

Move into the samples directory and run Bowtie 2.

$ bowtie2 -x RubriBowtie2 -1 GW1.fq\_1 -2 GW1.fq\_2 -S GW1.fq.sam -I 200 -X 800 -p 2 --no-unal --reorder

The -x option sets the reference index, -1 is the first set of paired reads, -2 is the second set of paired reads, -S gives the name of the sam file to be created, -I is the minimum insert size, -X is the maximum, --no-unal discards reads that did not align from the sam file, --reorder reorders the samples into numeric order, --no-head does not add the sam header to the top of the generated sam file.

This needs to be repeated for every read. In order to make this easier, we have included a script Bowtie2ScriptPrint.pl that will take a list of names and create the Bowtie 2 commands. The Names.txt file already contains the names of the files that we will be using. These can be easily added to a shell script that will run all the Bowtie 2 alignments. A shell script is simply a small program with a list of UNIX commands that can be used on the command line and will run the commands within the shell script.

$ perl Bowtie2ScriptPrint.pl >> RunBowtie.sh

$ chmod u+x RunBowtie.sh

$ ./RunBowtie.sh

The chmod step changes the permissions on the shell script to allow it to be executed.

This script can be run on the worker nodes using qsub.

Simply change Bowtie2.q to include your current directory. Then submit the job to qsub.

$ qsub Bowtie2.q

Setting up the Database and results folder

The Stacks Pipeline does most of the analysis and data curation using a relational database. The database manager MySQL is an open-source relational database manager and uses SQL. SQL or Structured Query Language is a programming language designed for use on databases. In Stacks, the database and SQL queries (commands sent to the database) are already implemented, but the database that Stacks uses must be created and configured. The directory where Stacks will store the results must also be created. Note that all commands in SQL must end with a semi-colon “;”.

1) Creating the Database

Use the mysql username and password, not the username and password used to log onto the Juniata HHMI cluster.

$ mysql -u sickler -p

Enter Password:JUNIATAsql

$ CREATE DATABASE StacksYOURUSERNAME;

#NOTE: Replace YOURUSERNAME with the username that you used to log on to Juniata’ HHMI Cluster.

$exit;

Next, configure the database for use by the Stacks Pipeline.

The Stacks Pipeline has a script that will configure the database you just created. You simply need to tell MySQL the database to use and what script to run on that database.

2) Configuring the Database

$ mysql -u sickler -p StacksYOURUSERNAME < /share/apps/Installs/Sickler/stacks-1.04/share/stacks/sql/stacks.sql

Enter Password:JUNIATAsql

If there was an error, a message describing it will appear on the screen. If not, then there will be a new command prompt.

3) Creating Results Directory

$ mkdir StacksResults

Running Stacks

Along with the database and output directory, the Stacks Pipeline also needs a population map that describes which individuals are in which population.

1) Making the Population Map

The Population Map contains “Sample Name” (tab) “Population Number”. The numbers do not need to be in sequential order as long as they are all integers. However, conventionally, the populations are usually numbered 1 .. n.

$ ls \*.sam >> PopulationMap

This will list all of the file names that end in “.sam” and create a new file called PopulationMap. Next it is necessary to edit the PopulationMap file using nano. First delete the “.sam” ending from each sample (but not the .fq). Then add a tab after each sample name and add the population number (the same as listed above in the file, either a 1 or 2).

2) Running Stacks

Part of the Stacks command is a list of each individual file that will be used as part of Stacks. It is generally easier to **make a small shell script** that can be easily read, edited, and saved and can also run Stacks. We will name our shell script “RunStacks.sh”, and get started by inputting the names of the same files:

$ ls \*.sam >> RunStacks.sh

This will then add the sam files to a new file RunStacks.sh which you then need to edit into the script that will run Stacks. Edit RunStacks.sh to contain the contents below:

­­­­­­­­­­­­­­­­­­#!/bin/bash

$ref\_map.pl -o StacksResults -T 2 -O PopulationMap -B StacksYOURUSERNAME -b 1 –d \

-s GW1.fq.sam \

-s GW2.fq.sam \

-s GW3.fq.sam \

-s GW4.fq.sam \

-s GW5.fq.sam \

-s GW6.fq.sam \

-s GW9.fq.sam \

-s GW11.fq.sam

NOTES ON THE FILE:

* All UNIX commands are only one line, which may make the command string longer than the screen can normally show which makes the command difficult to read. To make the command easier to read, you can add a slash (\) to the end of a line and the computer will read any lines ending in a slash and the first line without a slash after them as one line.
* You added “–d ” to the end of the first line, before the first “\”. Calling the “-d ” option will make Stacks do a dry run. It will not run any data, but will tell you if something is wrong. You may not have included an individual in the Population Map or may have misspelled something. If there are no errors, you can remove the “–d ” and run Stacks.
* Note that to make this command an executable script, it will also need to be saved into a file and you will need to add “#! /bin/bash” as the first line of the file.

Now make the file executable as above and run it. Remember to delete -d\ if the data passes muster.

$ chmod u+x RunStacks.sh

$ ./RunStacks.sh

ref\_map.pl is a Perl script that controls the Stacks Pipeline to use reads that have already been mapped to a reference genome. The -o option is the path to the output directory, the -T option is the number of processors to use (we are using 2 for now), the -B option sets the database to be used, the -b option sets the number of the batch that is being used, the -s option must appear before every individual being analyzed and specifies the individual. Stacks can be run with modified parameters or inputs using the same database if they are in a different batch.

Populations (for extra information and or if there is not enough memory to run it on the head node)

Populations is the final part of the Stacks Pipeline and deals with how to extract data and what output formats to use as well as performing statistical analysis.

Occasionally, the head node does not have enough memory to run populations, so the worker nodes must be used. Since Stacks uses MySQL, the main part of Stacks must be run on the head node. Populations does not rely on MySQL, so it can be run using the worker nodes.

1) Run Populations to generate summary statistics.

$ populations -b 1 -P StacksResults -s -t 5 -M PopulationMap –k --structure --vcf

The -b option specifies the batch to use, the -P option is the path to the output directory to use, the -s option tells populations to output different files that can be implemented into an SQL database or for later analysis, -t sets the number of processors to you, -M specifies the population map, and -k uses kernel smoothing for statistical calculations. –structure generates a structure output file, and –vcf generates a vcf output file.

# Examining the results

The resulting files of the STACKS pipeline are very large. To examine the properties of these files and investigate how well your run went, UNIX and the program R-studio can be used to great effect. The main files we want to examine are entitled batch\_1.sumstats.tsv and batch\_1.fst\_1-2.tsv.

1. A description of the files are at: <http://creskolab.uoregon.edu/stacks/manual/#pfiles>. Please take a moment to examine the file structure.
2. Use nano to open the sumstats file. Delete the first few lines at the top containing individual IDs.
3. Compress the file using:

$gzip filename

1. Transfer the file to your computer as normal (using psFTP or scp)

# Launch Rstudio.

1. Import the dataset as a text file using “Import Dataset” icon in the upper right screen. The data has headers and is tab delimited.
2. Go to the menu item File/Open file and load the R-script that was provided to you.
3. To show how the various columns were imported, use, change the “Name of Dataframe” to your name below:

>str(Name of Dataframe)

Notice that spaces have been converted to “.” marks in headers.

1. To print to the console (bottom left screen in R studio) the first few lines of each column in a dataset use:

>head(Name of Dataframe)

You can also do this by selecting the name of the file in the workspace to view your data.

1. Use getwd() to see where your data is being saved to by default. You can change the location using setwd().
2. Our plan will be to merge the two datasets in a way that allows us to draw graphs on the data, as well as investigate effects of sample size (number of individuals supporting a SNP) on stacks divergence statistics. A limitation of the stacks out files is that divergence data (Fst) is in one file, and sample size (N) in another file, so we need to combine information from the two files. First we will make a common identifier for each SNP in the dataset in both the diversity and divergence files by concatenating the Chromosome, chromosome location, and SNP location into a new vector in the fst dataset called (Chr\_Locus\_BP). Then we’ll split the diversity file by population. Then we’ll merge the two population and Fst files to have one row of information per SNP.

To prepare the data for the merge, we need to have a uniquely named column in common between each dataset. We’ll use “fst” and “div” as the name of your fst and diversity files, respectively.

>fst$Chr\_Locus\_BP<-with(fst, paste(Chr,Locus.ID,BP,sep="\_"))

>div$Chr\_Locus\_BP<-with(div, paste(Chr,Locus.ID,BP,sep="\_"))

1. We will split the diversity file into a Pop1 and Pop2 file, using a “subset” function. We can subset certain rows and columns using: dataframe [rows,columns]. Below, from the “div” dataframe, we take rows with Pop.ID of 1 (div$Pop.ID==1)and all columns (nothing after the comma), and put them into a new dataframe called Pop1\_sumstat.

>Pop1\_sumstat<-div[div$Pop.ID==1,]

8. Do the same for population 2:

>Pop2\_sumstat<-div [div$Pop.ID==2,]

9. Now merge the two Pop files by:

>merged2<-merge(Pop1\_sumstat,Pop2\_sumstat,by="Chr\_Locus\_BP")

10. Now merge the diversity and divergence files using:

>merged3<-merge(merged2,fst,by="Chr\_Locus\_BP")

Note that the default style of merge is the intersection of the two data sets, so non-matching rows are excluded. By setting the option “,all=TRUE” one could keep the union of the two data sets.

11. Create a new variable containing the minimum of the two sample sizes from the two populations for each SNP.

>merged3$minN<-with(merged3,pmin(N.x,N.y))

Optional: Remove the original files from memory

>rm(fst)

>rm(div)

12. Now you can start graphing variable and making summary tables as you want. The following will provide a list of summary statistics:

>summary(merged3$Fst)

>summary(merged3$Fis.x)

>summary(merged3$Fis.y)

>summary(merged3$Obs.Het.x)

>summary(merged3$Exp.Het.x)

>summary(merged3$Obs.Het.y)

>summary(merged3$Exp.Het.y)

The following will produce a histogram of the vectors:

>hist(merged3$Fst)

>hist(merged3$Fis.x)

>hist(merged3$Fis.y)

>hist(merged3$Obs.Het.x)

>hist(merged3$Obs.Het.y)

The following will produce a set of boxplots showing how Fst varied with minimum sample size:

boxplot(Fst~minN,data=merged3)

boxplot(Fis.x~minN,data=merged3)

boxplot(Fis.y~minN,data=merged3)

To check for the number of distinct values in a column use (length(unique(vectorName))):

length(unique(merged3$Chr.x))

length(unique(merged3$Locus.ID.x))

length(unique(merged3$Chr\_Locus\_BP))

In this example, the first time a value appears, its value is copied into memory, and length determines the number of unique values.

# VARSCAN

This is an alternative SNP caller from BOWTIE files, which can also call indels.

1. From your 8\_stacks directory, move into your Samples directory. Convert the SAM files to binary (BAM) format

$module load samtools1.19

$samtools view -bS -o GW1.fq.bam GW1.fq.sam

1. Repeat for each sam file. Use up arrow to pull up the old command, and change the number manually.
2. Next, we sort the BAM file, in preparation for SNP calling. The program will add the suffix ".bam" to the sorted files as a service for you.

$samtools sort GW1.fq.bam GW1.fq.sorted

1. Repeat for each bam file
2. Move the sorted.bam files to your VarScan directory

$mv \*.sorted.bam ../VarScan

1. Move into the varscan directory where you will find the fish genome rubriAll.fasta. Call SNPS using mpileup as follows:

$samtools mpileup -f RubriAll.fasta GW\*.fq.sorted.bam > tempData.mpileup

1. The reference genome file is specified by -f. Below we run varscan to filter snps and indels for significance.

$java -jar /share/apps/Installs/walls/VarScan.v2.3.6.jar mpileup2snp tempData.mpileup -p-value 0.01 > tempVarScanSNP.txt

$java -jar /share/apps/Installs/walls/VarScan.v2.3.6.jar mpileup2indel tempData.mpileup -p-value 0.01 > tempVarScanINDEL.txt

The manual is at: <http://varscan.sourceforge.net/using-varscan.html>. To understand the output file, go to the link, scroll down to mpileup2snp and go to OUTPUT.

# ASSESSMENT

1.Repeat the analyses above but after filtering the data for rows containing a minimum sample size of 10 (see below for code). How did filtering results for comparisons of size at least 10 change the number of outliers in FST, or change means?

>merged3N<-merged3[merged3$minN>=10,]

2.Filter the main and size-filtered combined dataset for Fst values > 0.9 and draw a histogram for each.

3. Compare the number of SNPs called between Stacks and VarScan. What could account for the differences?

4. Build a script for the “samtools view” commands like you did above for running stacks.

# References

Bowtie 2 Manual: <http://bowtie-bio.sourceforge.net/bowtie2/manual.shtml>

Stacks Manual: <http://creskolab.uoregon.edu/stacks/manual/>

Populations Manual: <http://creskolab.uoregon.edu/stacks/comp/populations.php>

J. Catchen, P. Hohenlohe, S. Bassham, A. Amores, and W. Cresko. *Stacks: an analysis tool set for population genomics*. **Molecular Ecology**. 2013.

<http://onlinelibrary.wiley.com/doi/10.1111/mec.12354/abstract;jsessionid=0CA96F04838EC77C996C263CB7ACB70F.f01t03>

J. Catchen, A. Amores, P. Hohenlohe, W. Cresko, and J. Postlethwait. *Stacks: building and genotyping loci de novo from short-read sequences*. **G3: Genes, Genomes, Genetics**, 1:171-182, 2011.

<http://g3journal.org/content/1/3/171>

Langmead B, Salzberg SL. Fast gapped-read alignment with Bowtie2. Nat Methods. 2012;14:357–359. doi: 10.1038/nmeth.1923. <http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3322381/>

APPENDIX A: Atmosphere Instructions

Obtain your own Atmosphere account: (should be completed before the workshop)

1. [Sign up for an iPlant account](https://pods.iplantcollaborative.org/wiki/display/atmman/Requesting+Access+to+Atmosphere) and [request access to Atmosphere](https://pods.iplantcollaborative.org/wiki/display/DEmanual/Adding+an+App+or+Service). You may want to mention your affiliation with GCAT-SEEK. It usually takes about 24 hours for the account to be granted. You will receive emails confirming it.
2. From the main [iPlant website](http://www.iplantcollaborative.org/), click the Atmosphere **Login** icon and enter your iPlant username and password.

All users begin with 168 AUs (AU=1 processor, 1 hr), enough time to continuously run a 1-vCPU instance for 1 week. Users who exceed their allocation will have their instances suspended until their allocation is refreshed. Additional allocation can be obtained using the **Request more resources** form within Atmosphere.

# General information links

[**Getting Started with iPlant**](http://www.iplantcollaborative.org/learning-center/get-started)

[**Data Store Information**](http://www.iplantcollaborative.org/ci/data-store)*Hint: Once you log into the Data Store or DE, if the data window seems to hang, click the x in the corner of it and reopen it with the orange Data icon with a book on it in the upper left.*

[**Discovery Environment**](http://www.iplantcollaborative.org/ci/discovery-environment)

[**Atmosphere Information**](http://www.iplantcollaborative.org/ci/atmosphere)**,** [**Manual**](https://pods.iplantcollaborative.org/wiki/display/atmman/Atmosphere+Manual+Table+of+Contents)

Accessing Data Store ([iDrop & iCommands)](https://pods.iplantcollaborative.org/wiki/display/start/Data+Store+Quick+Start#DataStoreQuickStart-ConnectingwithiCommands)

# General Use Information

You will be given a temporary iPlant username and password for the workshop. This account has already been given access to the Discovery Environment and Atmosphere. If you are already signed into iPlant with your personal account, sign out. Login to the [iPlant website](http://www.iplantcollaborative.org/) using the provided username/password.

There will be four instances of Atmosphere in the left column under My Instance. On the top right, My Resource Usage displays graphs of how many of your allotted processors and storage space is being used. If you click on the instance title, information about the instance will appear on the right and the Resource Usage will display the usage of that instance darker than the overall account usage. The most important piece of information from this screen to note now is the instance IP address which appears under the instance name.

To access your instance, either use the IP address to ssh to it or VNC. You can [download](http://www.realvnc.com/download/viewer/) the VNC viewer or [download Putty](http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html) to use ssh. VNC is a graphical user interface however to run the programs, you will use the command line from within it, the same as using ssh for access. *(There are tabs on Atmosphere for ssh, see below, & VNC, however; those connections through Atmosphere do not always succeed. If the session connects, it is perfectly alright to use it although copy/paste do not work from outside of Atmosphere into the ssh window.)* Assuming PuTTY use, double click the exe and then enter the instance IP address in the box labelled “Host name (or IP address)”. Change nothing else and click Open. Putty may give a warning, proceed by clicking Yes. Use your workshop iPlant account username and password to login.

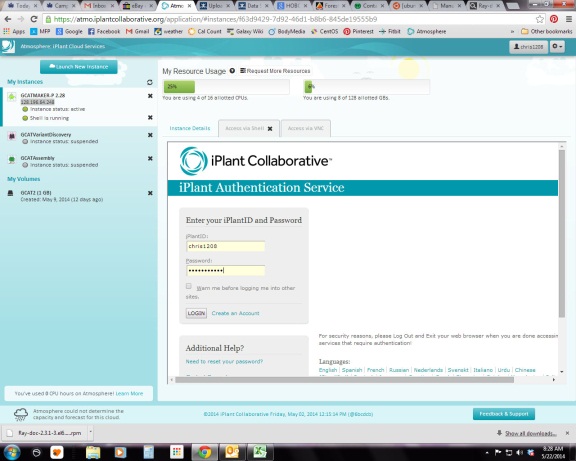


Figure 2 Atmosphere Instance SSH tab

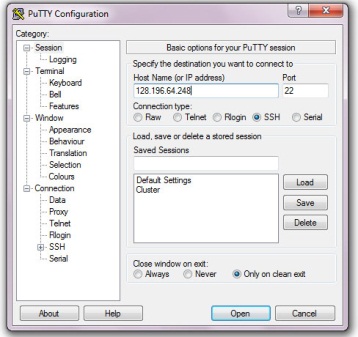


Figure 3 PuTTY Configuration Window (ssh)

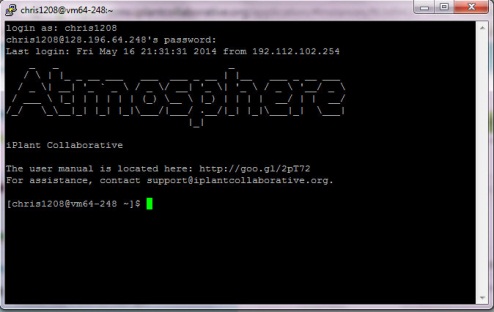


Figure 4 Command line after successful login.

Upon successful login, iniate iCommands for accessing data stored on the Data Store:

( see [Uploading Data to the Discovery E](https://pods.iplantcollaborative.org/wiki/display/DEmanual/Uploading+and+Importing+Data+Files+and+Folders+using+the+DE)nvironment for more information.)

Type: ($ indicates you are at the command line, do not type this character)

$iinit

The following line will appear:

One or more fields in your iRODS environment file (.irodsEnv) are

missing; please enter them.

Enter the following bold type as prompted (you can copy this text and right click to paste it into the command line.)

Enter the host name (DNS) of the server to connect to:**data.iplantcollaborative.org**

Enter the port number:**1247**

Enter your irods user name:<your username>

Enter your irods zone:**iplant**

Enter your irods password:<your password>

Once iCommands have been loaded, $ils will list the contents of your Data Store rather than the local current directory. Typing $ls will still show the contents of the local current directory. To upload files to the instance, use *iget* <filename>, *iput* will download back to the data store. Note that sudo must be used outside of your home directory (for instance to copy a file to /usr/lib.)

**It is important to put any files (data) that has been generated by your work on Atmosphere back to the Data Store before terminating the instance. All data on Atmosphere will be lost when the instance is terminated.**

# PREINSTALLED WORKSHOP INSTANCES

(follow instructions above to ssh to instance and load iCommands with $ iinit. Enter all path staments at the command line. The path statements can be copied from this document and pasted to the command line.)

## GCATAssembly1\_CW

$ export PATH=/usr/bin/:$PATH

$ export PATH=/usr/bin/jelly/bin/:$PATH

$ export PATH=/usr/bin/Quake/bin/:$PATH

$ export PATH=/usr/bin/SOAP/SOAPdenovo-V2.04/bin/:$PATH

$ export PATH=/usr/bin/SOAP/SOAPec\_v2.01/bin/:$PATH

Programs loaded:

* + - Jellyfish 1.11.11 (<http://www.cbcb.umd.edu/software/jellyfish/> )
    - Quake (<http://www.cbcb.umd.edu/software/quake/> )
    - SOAPdenovo2 (<http://soap.genomics.org.cn/soapdenovo.html> )

R (<http://www.r-project.org/> )

## GCATMAKER-P

$ export PATH=/opt/maker/bin/:$PATH

$ export ZOE=/opt/maker/exe/snap

$ export AUGUSTUS\_CONFIG\_PATH=/opt/maker/exe/augustus/config

Software loaded:

* MAKER-P [Atmosphere Maker tutorial](https://pods.iplantcollaborative.org/wiki/display/sciplant/MAKER-P+Atmosphere+Tutorial) (this tutorial demos running Maker using provided data)

Set up a MAKER-P run.  Create a working directory called "maker\_run" using the **mkdir** command and use **cd** to move into that directory:

$ mkdir maker\_run

$ cd maker\_run

Create another directory for your data (“test\_data” here) with mkdir. Use cd to move into the directory. From within your data directory, run $ils to see the files in your Data Storage. Any other directories that you have been given shared access to will also be available. Your instructor will give you the path to the file(s) you need. Type:

$ ils

$ iget <path/to/filenames>

If the files are in a tar ball, type the following to untar them:

$ tar -xvzf <filename.gz>

Save the data files generated to your own datastore. Using iput will copy files to your home directory on the Data Store (unless you have used icd to change directories within the Data Store.)

$ iput <filename>

## GCATVariantDiscovery\_CW:

$ export PATH=/usr/bin/:$PATH

Programs loaded:

* Stacks (<http://creskolab.uoregon.edu/stacks/manual/> )
* VarScan (<http://varscan.sourceforge.net/using-varscan.html> )
* R (<http://www.r-project.org/> )

# DISCOVERY ENVIRONMENT APPS

Click the middle icon on the left labelled “APPS”. Type the name of a program in the search bar to find apps of interest. Double click the App name to launch it. These are a few of the apps available for software you may use at some point.

ALLPATHS-LG v44837 <https://pods.iplantcollaborative.org/wiki/display/DEapps/AllpathsLG+44837>

Ray 2.2 <https://pods.iplantcollaborative.org/wiki/display/DEapps/Ray-2.2.0>

SOAPdenovo 2.04 <https://pods.iplantcollaborative.org/wiki/display/DEapps/Soapdenovo-2.04>

GapCloser <https://pods.iplantcollaborative.org/wiki/display/DEapps/Soapdenovo-2.04>

Appendix B: Copy files from Unix cluster to your desktop

For mac go to your mac terminal and use scp to copy to your desktop.

$scp testScaf500.all.gff [username@192.112.102.21:/path](mailto:username@192.112.102.21:/path)

For windows, double-click on psFTP on your computer and connect to the Juniata-HHMI cluster:

psftp>open username@192.112.102.21

Notice that your working directory is shown. Next transfer the file using the following command. The file will get transferred to the same folder where psFTP was located.

psftp>get path/testScaf500.all.gff

Appendix C: A few regular expressions

|  |  |
| --- | --- |
| \* | Wildcard, any character |
| . | Any character other than new line (\n) |
| \t | Tab |
| \s | Whitespace (space, Tab, new line) |
| ^ | Beginning of line |
| $ | End of line |
| \n | Carriage return |

<http://www.rexegg.com/regex-quickstart.html>