**INTRODUCTION TO DE NOVO TRANSCRIPTOME ASSEMBLY**

We are going to practice our transcriptome assembly with some of the STG data. We are going to use files that have been modified to be very small before we transition to working with the complete files. This way we can run through everything in a reasonable amount of time. Usually these steps (especially transcriptome construction) take MUCH longer.

Below are the names of the sample files. These are raw Illumina output files and are labeled by the read direction. PD\_R1.fastq.gz

PD\_R2.fastq.gz

GM\_R1.fastq.gz

GM\_R2.fastq.gz

Lets first have a look at what a raw Illumina looks like. Use the unix you learned to have a look at the first few lines of the “LookAtMe.fastq” file. The above files are gzipped (hence the .gz ending) and so you can’t easily look at them (but give it a try and see what happens).

Before we begin we will also need to load a couple of modules:

$ module load fastqc/0.11.5

$ module load gcc/4.7.1

$ module load trinityrnaseq/2.0.6

$ module load bwa/0.7.12

$ module load cd-hit/4.6.4

$ module load perl

$ module load bioperl/1.6.901

**Quality checking with FastQC**

Before we being working with our data we need to quality control the raw data. The first step is to inspect the quality of the reads. We will look at the average quality of each sample using a program called FastQC. To do so, run the following line of code:

$ fastqc PD\_R1.fastq.gz

This will generate a folder with a number of different types of output for us to download and inspect. After you run the program on this first sample, run it on the others as well.

Note that FastQC can use zipped files as input, which is very handy when you’re working with large files instead of the tiny files we are using here.

**Trimming with Trimmomatic**

To improve the quality of the raw reads, we will trim them using Trimmomatic. As with FastQC, this program can use zipped files as input. Note that this is not the only program that you can use to do this.

In addition to quality checking, Trimmomatic will remove adapter contamination. To do this we first need to put the adapters where the program can find them. We can do this by creating a softlink to the adapters from the folder we are working in. Modify the code below to do this:

Here is a general command line input:

$ ln –s /work/04148/efischer/programs/Trimmomatic-0.36/adapters/TruSeq3-PE.fa TruSeq3-PE.fa

Let’s look at this code one segment at a time:

$ java -jar /work/04148/efischer/programs/Trimmomatic-0.36/trimmomatic-0.36.jar PE -threads 32 -phred33 GM\_R1.fastq.gz GM\_R2.fastq.gz GM\_R1\_trim\_pair.fq.gz GM\_R1\_trim\_unpair.fq.gz GM\_R2\_trim\_pair.fq.gz GM\_R2\_trim\_unpair.fq.gz ILLUMINACLIP:TruSeq3-PE.fa:2:30:10:8:TRUE LEADING:25 TRAILING:25 SLIDINGWINDOW:1:25 MINLEN:36

* PE = paired end data. For single end data use SE.
* -phred33 = trim the reads using a phred score cut off of 33, meaning 99% confidence.
* The 4 fq file names are the output names. The output from Trimmomatic groups things as paired vs unpaired reads, and you’ll want to keep track of this. If you are working with zipped files the fq files should have the additional extension .gz because they are gzipped.
* ILLUMINACLIP options: the first file is the adapter.fa file, which contains a list of commonly used Illumina adapter sequences.
  + ILLUMINACLIP:<fastaWithAdaptersEtc>:<seed mismatches>:<palindrome clip threshold>:<simple clip threshold>
  + fastaWithAdaptersEtc: specifies the path to a fasta file containing all the adapters, PCR sequences etc.
  + seedMismatches: specifies the maximum mismatch count which will still allow a full match to be performed
  + palindromeClipThreshold: specifies how accurate the match between the two 'adapter ligated' reads must be for PE palindrome read alignment.
  + simpleClipThreshold: specifies how accurate the match between any adapter etc. sequence must be against a read.
* SLIDINGWINDOW:<windowSize>:<requiredQuality>;
  + windowSize: specifies the number of bases to average across;
  + requiredQuality: specifies the average quality required.
  + LEADING:<quality>; quality: Specifies the minimum quality required to keep a base.
  + TRAILING:<quality>; quality: Specifies the minimum quality required to keep a base.
  + CROP:<length>; length: The number of bases to keep, from the start of the read.
  + HEADCROP:<length>; length: The number of bases to remove from the start of the read.
  + MINLEN:<length>; length: Specifies the minimum length of reads to be kept.

Modify the code above as necessary and run it. Once complete, you should get the following output files:

GM\_R1\_pair.fq GM\_R1\_unpaired.fq

GM\_R2\_pair.fq GM\_R2\_unpaired.fqRepeat the above steps for the reads for the second set of samples (i.e. those that start with PD). After you are done, re-run the quality check with FASTQC on one set of the new, paired files to see how your read stats have changed.

**De novo transcriptome assembly**

Now that the reads are trimmed, we are ready for de novo assembly. We will use the program Trinity as an example here. This is not the only program! There are many programs with different pros and cons and more are continuing to be developed. Moreover, there are various downstream options to quality check, combine, filter, etc. assemblies. There is not a single right way to construct a transcriptome, but it is important to keep the trade-offs in mind and do the work to ensure you’re making informed decisions about what you do.

A few good things about Trinity are that it is open source (i.e. free), has good documentation, many useful downstream tools, a good help page, and is constantly being updated/improved. Check out the website to understand more of the options:

[github.com/trinityrnaseq/trinityrnaseq/wiki](http://github.com/trinityrnaseq/trinityrnaseq/wiki)

The first step is that we need to concatenate all the files so that we have a single file with all the R1 reads and a single file with all the R2 reads. We also need to unzip the files at this point:

$ cat \*R1\_trim\_pair.fq.gz > compiled\_R1.fq.gz

$ gunzip compiled\_R1.fq.gz

Here is a basic line of code for a Trinity run.

$ Trinity --seqType fq --left compiled\_R1.fq --right compiled\_R2.fq --SS\_lib\_type RF --min\_contig\_length 200 --JM 100G --output trinity --CPU 12 --full\_cleanup --inchworm\_cpu 12 --bflyHeapSpaceMax 20G --bflyCPU 8

Let’s take a fieldtrip to the website to understand what some of this means:

[github.com/trinityrnaseq/trinityrnaseq/wiki/Running%20Trinity](BUT,%20let’s%20take%20a%20fieldtrip%20to%20the%20website%20to%20understand%20what%20some%20of%20this%20means:%20github.com/trinityrnaseq/trinityrnaseq/wiki/Running%20Trinity)

We’re not actually going to run this code, because it will take too long (Trinity can run for hours to weeks depending on the amount of data you have). I’ve run it for you and the output of this file is "trinity.Trinity.fasta". Let’s have a look at the first few lines of this file to see what .fasta files look like.We can also take a look at some basic assembly characteristics by using a script that is part of the Trinity pipeline:

$ $TACC\_TRINITY\_DIR/util/TrinityStats.pl trinity.Trinity.fasta

What can you tell about this fasta file now?

**Cleaning up the raw assembly**The raw trinity assembly contains gene sequences as well as false contigs called assembly artifacts. We will now do a multi-step process that cleans up the raw trinity assembly. Here – as elsewhere – there are many options for clean up and quality checking (the trinity web page has a nice section with a range of additional suggestions).

We will first remap the raw read files to our filtered assembly using BWA and then generate a statistics file using eXpress. We are not doing these steps today because they take too long, but the code is below and I am giving you the final output.

The first step is to build an index of the raw trinity file that bwa can use in mapping:Next, we will remap the reads:

$ bwa mem trinity.Trinity.fasta compiled\_R1.fq compiled\_R2.fq > bwa\_all.sam

$ bwa index trinity.Trinity.fasta

After bwa is finished, we can now use the SAM file to generate a statistics file using eXpress.This will generate a file called results.xprs, that we will now use to filter out low confidence contigs. We will run a series of perl scripts that will allow us to make a fasta file containing contigs with an FPKM < 1 (FPKM = Fragments Per Kilobase of transcript per Million mapped reads).

$ express trinity.Trinity.fasta bwa\_all.sam

Run the following perl script, which will remove all lines with a value of less than 1 in column 11 (the FPKM column). Your output file contains contig stats that have an FPKM above 1.Now that you have filtered out the list of contigs with an FPKM below 1, you need a list of contig names that you want to keep (meaning you only want the list of contig names in column one).You now want to extract the “good” contigs from your list (those that survived the FPKM cutoff)from the raw trinity assembly. Make sure your raw trinity assembly is in the same folder as your express results and your fpkm.txt file from above.

$ perl -e ' $col=11; $limit=1; $count=0; while(<>) { s/\r?\n//; @F=split /\t/, $\_; if ($F[$col] > $limit) { $count++; print "$\_\n" } } warn "\nChose $count lines out of $..\n\n" ' results.xprs > fpkmfilter.txt

$ perl -e ' ($id,$fasta)=@ARGV; open(ID,$id); while (<ID>) { s/\r?\n//; /^>?(\S+)/; $ids{$1}++; } $num\_ids = keys %ids; open(F, $fasta); $s\_read = $s\_wrote = $print\_it = 0; while (<F>) { if (/^>(\S+)/) { $s\_read++; if ($ids{$1}) { $s\_wrote++; $print\_it = 1; delete $ids{$1} } else { $print\_it = 0 } }; if ($print\_it) { print $\_ } }; END { warn "Searched $s\_read FASTA records.\nFound $s\_wrote IDs out of $num\_ids in the ID list.\n" } ' fpkmkeep.txt trinity.Trinity.fasta > trinity\_fpkmfiltered.fasta

$ perl -e ' @cols=(1); while(<>) { s/\r?\n//; @F=split /\t/, $\_; print join("\t", @F[@cols]), "\n" } warn "\nChose columns ", join(", ", @cols), " for $. lines\n\n" ' fpkmfilter.txt > fpkmkeep.txt

You should now have a new fasta file that contains high confidence contigs. If you want to determine how many contigs were removed, you can run the following:

$ grep ">" -c trinity.Trinity.fasta

$ grep ">" -c trinity\_fpkmfiltered.fasta

We will now cluster contigs to combine separate contigs that have significant sequence overlap. Run the following line of code:

$ cd-hit-est -i trinity\_fpkmfiltered.fasta -o assembly\_cdhit98.fasta -c 0.98 -n 8

After clustering contigs, we will now remove all contigs smaller than 250 bp using a perl script. Note: you can change the –b option to use a cutoff of a different size, depending on your needs. To look at the statistics of your assembly you can use the same script we used before. This will provide you a nice comparison of how the assembly has changed post filtering:

$ perl /work/04148/efischer/scripts/binSeqsByLength.pl -b '250' assembly\_cdhit98.fasta

$ $TACC\_TRINITY\_DIR/util/TrinityStats.pl assembly\_cdhit98\_250\_or\_more.fasta

Congratulations! You now have a draft assembly of a transcriptome!