RNA-Seq Differential Analysis Pipeline Summary

Contents

List of Figures

This is a brief summary of the pipeline and tools I developed during my time in the MS program at UW -- Milwaukee.

I have 40 years of programming experience, 30 years in Unix systems management, and 20 years in scientific computing support, but my direct involvement in biology research is minimal, so I would love to hear from others about their experiences working with sequence data.

1 Clusters in Bioinformatics

Use of a cluster is not necessary for most differential expression analyses. A powerful workstation with several cores and enough RAM can complete a a typical analysis in a reasonable amount of time. In fact, researchers are likely to spend more time learning HPC and waiting for help from support staff than it would take to do the analysis on their own on a decent workstation.

There are some activities that are impractical without a cluster, such as de novo genome assembly, but most bioinformatics can be done on modern laptops and workstations.

A major goal of my research is to make DE analysis accessible to a broader population of research biologists, and part of that is demonstrating how to perform an analysis on a simple workstation. To achieve this, I have made it possible to install all of the software necessary for a DE analysis with ease, and run it with minimal Unix skills. Only basic Unix command-line and shell scripting are necessary. No Perl, Python, or R programming is required.

This repository contains both SLURM scheduler scripts for running on a cluster and scripts for running on a standalone Unix machine using **xargs** to run jobs in parallel. The **xargs** scripts are tested primarily on a Mac Mini m1.

2 Software Installation

All programs necessary for this pipeline can be installed on FreeBSD using **pkg install rna-seq**.

On any POSIX platform (BSD, Linux, macOS, SunOS, etc.), they can be installed using the **pkgsrc** rna-seq package (http://pkgsrc.org/-). Pkgsrc does not require administrator privileges, so it can be used on computers managed by your organization without submitting help requests.

Pkgsrc can be set up in about 10 minutes using **auto-pkgsrc-setup** (http://netbsd.org/~bacon/). Binary packages are available for some platforms (NetBSD, macOS, RHEL Linux). In this case, we can use **pkgin install rna-seq**. On most platforms, pkgsrc users install from source using **cd (prefix)/biology/rna-seq; bmake install**.

This repository includes a script that installs the software using FreeBSD ports or pkgsrc.

Programs used in this pipeline (and included in the rna-seq meta-packages) are described in the sections that follow.

3 Recompression

Recompressing raw data from gzip format to xz format generally reduces disk usage by about 40%. This reduced xenopus raw reads from 143 GB to 92 GB.

Compression with **xz** at standard compression levels is time-consuming, so I only do this for long-term data, not for intermediate results. Decompression with **unxz** is very fast, so reading xz-compressed files does not slow down the analysis.

For intermediate output files, I generally use **zstd**. This tool has pretty much obsoleted **gzip**, since it is both faster and produces better compression ratios.

All of the tools I developed can directly read and write files compressed with **gzip**, **bzip2**, **xz**, **lz4**, and **zstd**.

4 Raw File Aliases

Sequencing centers may require ridiculously compact filenames for the sequence files (are the running MS-DOS??). To avoid confusion and mistakes caused by cryptic filenames, I first create symbolic links with more descriptive names that are both easier for people to read and easier for shell scripts to parse.

For this study, I also reduced the time points to ranks 1 through 5. The axolotl and xenopus data have the same number of samples and time points, to standardizing the filenames allows the exact same analysis scripts to be used for both data sets. This saves a lot of time on bug fixes and documentation.

```
Symbolic link                       Raw filename from sequencing center
-------------------------------------------------------------------------
sample01-time1-rep1-R1.fastq.xz@ -> ../../../Raw/X1NaA/X1NaA_1.fq.xz
sample01-time1-rep1-R2.fastq.xz@ -> ../../../Raw/X1NaA/X1NaA_2.fq.xz
sample02-time1-rep2-R1.fastq.xz@ -> ../../../Raw/X2NaB/X2NaB_1.fq.xz
sample02-time1-rep2-R2.fastq.xz@ -> ../../../Raw/X2NaB/X2NaB_2.fq.xz
sample03-time1-rep3-R1.fastq.xz@ -> ../../../Raw/X3NaC/X3NaC_1.fq.xz
sample03-time1-rep3-R2.fastq.xz@ -> ../../../Raw/X3NaC/X3NaC_2.fq.xz
sample04-time2-rep1-R1.fastq.xz@ -> ../../../Raw/X4d7A/X4d7A_1.fq.xz
sample04-time2-rep1-R2.fastq.xz@ -> ../../../Raw/X4d7A/X4d7A_2.fq.xz
sample05-time2-rep2-R1.fastq.xz@ -> ../../../Raw/X5d7B/X5d7B_1.fq.xz
sample05-time2-rep2-R2.fastq.xz@ -> ../../../Raw/X5d7B/X5d7B_2.fq.xz
sample06-time2-rep3-R1.fastq.xz@ -> ../../../Raw/X6d7C/X6d7C_1.fq.xz
sample06-time2-rep3-R2.fastq.xz@ -> ../../../Raw/X6d7C/X6d7C_2.fq.xz
sample07-time3-rep1-R1.fastq.xz@ -> ../../../Raw/X7d12A/X7d12A_1.fq.xz
sample07-time3-rep1-R2.fastq.xz@ -> ../../../Raw/X7d12A/X7d12A_2.fq.xz
sample08-time3-rep2-R1.fastq.xz@ -> ../../../Raw/X8d12B/X8d12B_1.fq.xz
sample08-time3-rep2-R2.fastq.xz@ -> ../../../Raw/X8d12B/X8d12B_2.fq.xz
sample09-time3-rep3-R1.fastq.xz@ -> ../../../Raw/X9d12C/X9d12C_1.fq.xz
sample09-time3-rep3-R2.fastq.xz@ -> ../../../Raw/X9d12C/X9d12C_2.fq.xz
sample10-time4-rep1-R1.fastq.xz@ -> ../../../Raw/X10d18A/X10d18A_1.fq.xz
sample10-time4-rep1-R2.fastq.xz@ -> ../../../Raw/X10d18A/X10d18A_2.fq.xz
sample11-time4-rep2-R1.fastq.xz@ -> ../../../Raw/X11d18B/X11d18B_1.fq.xz
sample11-time4-rep2-R2.fastq.xz@ -> ../../../Raw/X11d18B/X11d18B_2.fq.xz
sample12-time4-rep3-R1.fastq.xz@ -> ../../../Raw/X12d18C/X12d18C_1.fq.xz
sample12-time4-rep3-R2.fastq.xz@ -> ../../../Raw/X12d18C/X12d18C_2.fq.xz
sample13-time5-rep1-R1.fastq.xz@ -> ../../../Raw/X13d27A/X13d27A_1.fq.xz
sample13-time5-rep1-R2.fastq.xz@ -> ../../../Raw/X13d27A/X13d27A_2.fq.xz
sample14-time5-rep2-R1.fastq.xz@ -> ../../../Raw/X14d27B/X14d27B_1.fq.xz
sample14-time5-rep2-R2.fastq.xz@ -> ../../../Raw/X14d27B/X14d27B_2.fq.xz
sample15-time5-rep3-R1.fastq.xz@ -> ../../../Raw/X15d27C/X15d27C_1.fq.xz
sample15-time5-rep3-R2.fastq.xz@ -> ../../../Raw/X15d27C/X15d27C_2.fq.xz
```

5 Pre-trim Quality Checks

The first analysis step is to run FastQC on the raw files, and MultiQC to combine the results for all samples into one interactive HTML report. A couple of example plots from the MultiQC report are shown here.

The read quality plot (axolotl data) shows that almost all of the reads are very high quality (A phred score of 20 means a 1/100 chance of a read error, 30 means a 1/1000 chance of error).

Figure 1: Read Quality

The adapter content plot shows that up to 2.5% of reads in some samples have adapter contamination near the 3' end.

Figure 2: Adapter Content

We can also run tools such as **blt fastx-stats** (from biolibc-tools (https://github.com/auerlab/biolibc-tools), which is installed by the rna-seq meta-packages. Biolibc-tools is a collection of simple tools I created to provide permanent solutions to numerous

simple problems that often present a nuisance to researchers. It can be thought of as "putty" to fill in small gaps in typical analyses.

```
# blt fastx-stats Results/01-organize/Raw-renamed/sample01-day00-rep1-R1.fastq.xz
Filename:           Results/01-organize/Raw-renamed/sample01-day00-rep1-R1.fastq.gz
Sequences:          57943764
Bases:              8691564600
Mean-length:        150.00
Standard-deviation: 0.00
Min-length:         150
Max-length:         150
A:                  2478136460 (28.51%)
C:                  1873856390 (21.56%)
G:                  1887781386 (21.72%)
T:                  2451663620 (28.21%)
N:                  126744 (0.00%)
GC:                 3761637776 (43.28%)
```

6 Trimming

Trimming is not really necessary for RNA-Seq differential analysis, since the aligners will deal with adapter contamination, poly-A tails, etc. (Liao, 2020, doi: 10.1093/nargab/lqaa068). However, trimming beforehand reduces the workload for subsequent analysis stages, and allows them to produce cleaner results.

While I was perfectly happy with **cutadapt**, I wrote my own tool for this, called **fastq-trim**, (https://github.com/auerlab/fastq-trim) just to see how much improvement was possible over a quality existing tool. As shown on the website, **fastq-trim** is about 2.5 times as fast as cutadapt and 4 times as fast as Trimmomatic, while also using far less memory than either of them.

It uses algorithms very similar to cutadapt and a more efficient implementation in C. Testing on a sample of 100,000 reads produced only 6 trimmed sequences that differed slightly from the cutadapt output.

Trimming the xenopus data took 1.5 hours using 15 cores (5 jobs at a time, 3 cores per job) on my personal cluster (10-year-old Dell PowerEdge R415 servers). Average time to trim an individual file was about 30 minutes. It also ran in about 1.5 hours on my Mac Mini M1, using 4 jobs at a time under xargs.

Output from one sample of xenopus data is below. Results for all other samples were similar.

```
*** FASTQ TRIM ***

  Minimum match:     3
  Minimum quality:   24
  Minimum length:    30
  Phred base:        33
  Adapter matching:  Smart
  Maximum mismatch:  10%
  Filename:          Results/01-organize/Raw-renamed/sample02-time1-rep2-R1.fastq.xz
  Filename:          Results/01-organize/Raw-renamed/sample02-time1-rep2-R2.fastq.xz
  Mode:              Paired
  Adapters:          AGATCGGAAGAG AGATCGGAAGAG



Reads:                             107099288
Reads with adapters:                 5150891 (4%)
Reads with Poly-As:                   349808 (0%)
Bases with Q < 24:                  60688727 (0%)
Reads with low Q bases removed:      5173865 (4%)
Reads < 30 bases after trimming:      124850 (0%)
Mean adapter position:                   134
Mean read length:                        150
```

```
Run time: 1337.33 real      1621.45 user       137.50 sys
```

7 Post-trim Quality Check

Running FastQC and MultiQC on the trimmed data should show improvements in many metrics. The plot below shows that most adapter content has been removed. FastQC may flag some sequences as adapters that the trimming tool wasn't even looking for, so don't expect to find zero contamination after trimming. And again, removing adapter contamination isn't even necessary for RNA-Seq differential analyses. We trim merely to get rid of most contamination in order to improve the efficiency of subsequent stages.

Figure 3: Adapter Contamination after Trimming

8 Reference Transcriptome and/or Genome

I align reads to both a transcriptome (using **kallisto**) and a genome (using **hisat2**), and then compare the results for validation.

Aligning to the genome also enables discovery of novel transcripts.

For the first run on the axolotl reads, I used transcriptome `AmexT_v47_cds.fa` and genome `AmexG_v6.0-DD.fa` from http://www.axolotl-omics.org as references.

For well-annotated organisms, we can also create a transcriptome using **gffread**, a GTF or GFF3 file, and a genome FASTA. Generating from the GTF/GFF3 ensures that all annotated sequences present in GTF/GFF3 are also present in the transcriptome. This is not always the case for downloaded cDNA transcriptomes.

9 Alignment to a Transcriptome

I use **kallisto** for this step. It is extremely fast (if not generating pseudobams or bootstrap estimates, which are usually not necessary). Kallisto does not generate SAM (sequence alignment map, or BAM, a compressed equivalent) files, like most aligners to a genome. Creating pseudobams was an option prior to kallisto 0.50.0. Now, we need another aligner in order to generate SAM/BAM files.

Kallisto also performs quantification, producing raw read counts for each transcript (used by differential analysis tools) and TPM (transcripts per million) for human-consumption. A sample of **kallisto** output is shown below.

```
target_id               length  eff_length  est_counts  tpm
AMEX60DD201000002.1;    2001    1737.38     97.1699     0.798628
AMEX60DD201000003.1;    2508    2244.38     250.115     1.59129
AMEX60DD301000003.2;    1623    1359.38     0           0
AMEX60DD201000004.1;    1692    1428.38     90.0984     0.900701
AMEX60DD201000005.2;    357     101.384     19          2.67603
```

The "eff_length" and "est_counts" columns are used by DA tools such as DESeq2, EdgeR, FASDA, and Sleuth, to compute fold-changes and P-values. TPM, RPKM, and FPKM are reductions of read count information that not useful to DA tools, and are mainly used to give readers a rough idea about mRNA abundance.

10 Quantification

Quantification is the process of counting the reads aligned to each feature of interest (genes or transcripts for RNA-Seq) and estimating relative mRNA abundance from these data. Normally, this is a separate step, but as mentioned above, **kallisto** does this for us automatically during the alignment process.

11 Differential Analysis

This is the most problematic stage of the analysis. Most existing tools (DESeq2, EdgeR, Sleuth, etc.) require the user to write R scripts that read the alignment / quantification output (which differs for different aligners), manipulate it into complex R data structures (called data frames), and use R library functions to generate the fold-changes and P-values used to identify significant changes in expression.

Installing and maintaining R packages is also notoriously problematic. The installations using **install.packages()** and **BiocManager::install()** often fail, and will almost certainly cease to function a few months down the road due to routine updates to the underlying OS and packages installed outside of R.

In addition to technical issues with R, the results very greatly across different DA tools. Note that 3 samples are not nearly enough to achieve satisfactory statistical power in any setting. Many DA tools use sophisticated statistical tricks to increase power at the cost of higher false discovery rates. From Li, et al:

" DESeq2 and edgeR had large discrepancies in the DEGs they identified on these datasets (Additional file 1: Fig. S1). In particular, 23.71% to 75% of the DEGs identified by DESeq2 were missed by edgeR. The most surprising result is from an immunotherapy dataset (including 51 pre-nivolumab and 58 on-nivolumab anti-PD-1 therapy patients) [8]: DESeq2 and edgeR had only an 8% overlap in the DEGs they identified (DESeq2 and edgeR identified 144 and 319 DEGs, respectively, with a union of 427 DEGs but only 36 DEGs in common). This phenomenon raised a critical question: did DESeq2 and edgeR reliably control their false discovery rates (FDRs) to the target 5% on this dataset? "

Lastly, R is an interpreted language, which is roughly 100 times slower than fully compiled languages such as C, C++, or Fortran. This is not a problem for differential expression analyses using a small number of replicates, but it's a big problem for population studies (Li, et al, 2022, https://doi.org/10.1186/s13059-022-02648-4).

Li, et al also reported poor performance for large sample sizes (the performance issue is not relevant to our 3-sample study).

To alleviate many of these issues, I developed a new differential analysis tool called FASDA (https://github.com/auerlab/fasda), which is written entirely in C, is easily installed and updated using highly-evolved and reliable package managers, and is installed by rna-seq meta-packages mentioned above.

It processes kallisto output directly, so there is no need for R programming or knowledge of data frames. It computes *exact* P-values for low sample counts, and uses the Mann-Whitney U test (a.k.a. Wilcoxon rank-sum test) for sample counts of 8 or more. Both of these methods control false discovery.

Note Exact P-values are computed by literally generating all possible combinations of read counts and counting combinations with fold-changes as least as extreme as observed. Hence, such P-values are not estimates, like those produced by other tools.

Processing **kallisto** output with **fasda** requires only two commands, e.g.:

```
# Generate normalized counts from all samples under the same condition
# This uses the median-ratios normalization (MRN) method
fasda normalize --output normalized-counts-condition1.tsv \
    sample*-condition1-rep*/abundance.tsv

# Compute fold-changes and exact or Mann-Whitney P-values
fasda fold-change --output FCs-c1-c2.txt \
    normalized-counts-condition1.tsv \
    normalized-counts-condition2.tsv
```

Example output is shown below (MNC = mean normalized count, SD/C1 = standard deviation for MNC1 / MNC1, %Agr = how many samples agree on the direction of the change, FC = fold-change, P-val = exact P-values).

```
Feature                 MNC1    MNC2  SD/C1  SD/C2  %Agr  FC 1-2  P-val
AMEX60DD201000002.1;    99.1   102.2    0.2    0.3    67    1.03  0.79163
AMEX60DD201000003.1;   327.7   445.6    0.3    0.2    67    1.36  0.19409
AMEX60DD301000003.2;     0.0     0.0    0.0    0.0   100       *  1.00000
AMEX60DD201000004.1;    95.2    88.1    0.3    0.3    67    0.93  0.73842
AMEX60DD201000005.2;    23.5    16.3    0.6    0.3    67    0.69  0.49557
AMEX60DD201000006.1;   382.0   496.6    0.4    0.3    67    1.30  0.29852
```

12 Alignment to a Genome

I also align to the genome, mainly for comparison and validation of the kallisto analysis. This requires a splice-aware aligner, of which there are three in the mainstream:

- Hisat2, evolved from TopHat, is the fastest (Musich, 2021, doi: 10.3389/fpls.2021.657240) and most memory-efficient (except for indexing very large genomes, like axolotl, where it required over 70 GB). Aligning the reads has much more modest memory requirements.
- Salmon is notoriously difficult to build, so the only viable option for most researchers is to use the precompiled executables provided by the developers. Downloading and running executables this way is considered a very bad practice in I.T. due to the risk of viruses, Trojan horses, etc. It's better to install executables from a more trustworthy source, such as Debian packages, FreeBSD ports, pkgsrc, etc.

  Memory requirements are enormous, which means it will not run on a typical PC and cannot utilize all the cores on a typical HPC node, so the alignment will take much longer than necessary.

  The project currently has 290 open issues, which is not an indication of high code quality and active maintenance. https://github.com/-COMBINE-lab/salmon.
- STAR has the same issues with memory use and code quality as Salmon, but worse: 690 open issues at the time of this writing. https://github.com/alexdobin/STAR.

I experimented with all three of these aligners and now only use hisat2.

Unlike kallisto, these aligners do not perform quantification. FASDA performs quantification on the SAM/BAM/CRAM output of these aligners, generating an output file in the same format as kallisto's abundance.tsv, so that the normalization and DA stages are exactly the same regardless of the aligner used.

By default, FASDA uses **stringtie**, a highly-evolved abundance calculator often used on **hisat2** output. FASDA runs stringtie to compute abundances and then reformats the output into a kallisto-style abundance.tsv file.