**Using the DuffyNGS Pipeline**

A rough README description of what is needed to process NextGen data, with sections on setting up a computation environment, the basics of alignment and QC, RNA-seq transcriptomes and differential expression, SNP detection in WGS datasets, and extra sections covering advanced peak picking of ChIP-seq and RIP-seq datasets.

**Part 1: Getting Started**

**1.1. Hardware and Software Basic Requirements**

First off, a computing environment that supports the R language, interactive graphics, and enough hardware memory and disk for NextGen data. A Unix/Linux based system is required.

A) R version 2.9 or newer.

B) Support for X11 graphics. An easy check, once R is installed and running, is the R command: demo(graphics). It should open a plotting window, and generate ~10 different plot images using various fonts.

C) Compute hardware with access to at least 20-120GB of RAM (lower limit for bacteria, upper limit for mammalian genomes), 4-16 cores, and 1TB+ disk. Typical hardware may have a job scheduling queue system like SGE, SLURM, etc. Typical DuffyNGS tasks may involve both batch oriented jobs submitted to the scheduler and interactive X11 sessions. Support for both is required.

**1.2. Required Software Tools**

Next step, gathering the required open source software tools that we need to process NextGen data.

A) The DuffyTools and DuffyNGS R packages. Current version is 1.7.0, both tend to be constantly updated, shared as tarball .gz packages. Install the latest version whenever updated, via either ‘make’ from the Unix command line, or via: install.packages( …, repos=NULL, …) from inside R. Once installed, test that the installation worked with R function: library(DuffyTools). You should see several cryptic species IDs as the package loads the family of genome annotations. Recently, the packages have been uploaded to GitHub under: <https://github.com/robertdouglasmorrison>

As to a brief delineation of how functionality is divided between the two packages, the DuffyTools package contains genome annotations, general gene expression and bioinformatics tools, file manipulation tools, etc.; while DuffyNGS contains higher level Next Gen Sequencing pipelines and tools for manipulating NextGen data files.

B) Other R packages used by DuffyNGS. Other needed packages can be verified and/or automatically installed by the DuffyTools R function: checkDuffyPackages(). It will query the current R installation, and try to install any missing needed packages.

C) Bowtie2 aligner, from Johns Hopkins.

[http://bowtie-bio.sourceforge.net/bowtie2/index.shtml](file:///\\cidr\(http\::bowtie-bio.sourceforge.net:bowtie2:index.shtml))

Version 2.9.0 or older is preferred. Had some negative performance issues with versions 2.3.0 and newer 3.x.x – waiting on fix from authors. Bowtie1 is also used for building some of the “genome detectability” data objects, but should not be needed by general users.

D) SAMTOOLS utilities for SAM/BAM format files.

<https://sourceforge.net/projects/samtools/files/samtools/>

Version 1.3.1 or newer. There are some fatal bugs in versions prior to 1.3, that cause errors in BAM extraction and SNP calling.

E) Others, not strictly needed for RNA-seq or ChIP-seq, but commonly used by the DuffyNGS tools, include: VELVET, Python, MAFFT, BLAST and some Blast databases like ‘NT’, seq2HLA, etc. Let’s worry about those if/when we need to.

F) A new function has been added that tries to find and report the version numbers of all third party programs used by the DuffyNGS pipeline steps. Very useful to run it as a sanity check. Run checkDuffyPrograms(). It will make use of the Sys.which() R function and the Unix environment variables, to try to call all the third party programs, and report their version number and full pathnames.

**1.3. Setting up a Disk Folder Environment**

The pipeline uses and generates many files during the processing of a sample. A well organized file system layout helps keep us all sane. Typically, processing of all dataset families happens under one master folder per user/lab, so required common data (like genomes and Bowtie indexes) can be shared, and results for each user and dataset family can be kept separate. A typical setup consists of the following folders:

A) bin - one shared subfolder of commonly used executable programs, shell scripts, and R scripts; with read & execute privileges for everyone in the group.

B) Bowtie2Indexes - one shared subfolder of Bowtie genomic index files (as made by ‘bowtie2-build’), and the genomes etc. used to create them. We typically have a subfolder for each “target organism”, holding the scripts needed to build/rebuild the indexes for that target. This folder is populated manually, and only gets updated when new target genomes are needed.

C) A separate folder for each family of related datasets to be treated as one “Experiment”, typically named to briefly describe the experiment and lead scientist. In general, datasets that will be compared against each other will want to be kept together as one experiment, while datasets that won’t be compared usually will be kept in separate folders. For example, ChIP data and RNA-seq data from the same experiment will typically be processed separately and would benefit from separate folders. Similarly, if separate experiments get multiplexed for sequencing to save money, they most likely should be divided into separate folders even though the raw FASTQ data came back as one big folder. To repeat, the general rule is that all datasets that will be compared against each other live inside one experiment folder.

Most of the remaining sections of this README will focus on setting up the context and data to process one experiment folder of datasets.

**Part 2: Setting Up to Process an Experiment**

In the DuffyNGS nomenclature, an experiment is a family of raw NGS datasets that will be processed similarly and then compared/contrasted within this family of datasets. Each experiment generates a rather large number of subfolders and files of results, so we tend to relegate experiments to separate folders to stay sane and prevent filename collisions that might overwrite important data.

**2.1. Expected Files:**

There are a small number of expected files in each experiment folder that will control how the datasets are processed:

A) Annotation.txt: The file that defines the SampleIDs, the filenames for the FASTQ files, and other needed details for each dataset. One row for each sample.

Note on choosing SampleIDs: the pipeline tools make heavy use of the SampleID, as a file name prefix, subfolder name, and R data column name. Try to choose short yet clear descriptions, and avoid any blanks or special characters that are not allowed in filenames, R variables, R data column names, etc.

B) Options.txt: The file that sets runtime settings for various pipeline tasks. It contains the name of the target organism(s) for the experiment, the names of the Bowtie2 indexes, and many other runtime settings.

C) pipelineSetup.R (or similar): An small R script, typically run via source(“pipelineSetup.R”) as the very first command in every R session/job, that loads the DuffyNGS library, sets the default SpeciesID, and loads the annotation file. It may optionally contain a few wrapper functions that do routine pipeline steps like differential expression, etc.

D) multiPipeline.sh (or similar): A Unix shell script for submitting samples to the compute cluster for batch job processing.

E) (ChIP-seq & RIP-seq only) multiPeakPick.sh, multiPeakDraw.sh, peakpick.sh, peakdraw.sh: additional shell scripts for the peak picking and peak drawing pipeline steps. These steps may need to be repeated to allow inter-sample scaling, after all samples have been processed.

**2.2. Prepping the raw FASTQ data:**

The raw FASTQ files for each experiment may come in various layouts, formats, and perhaps as multiplexed separate files. The DuffyNGS pipeline mostly expects either one unpaired or exactly 2 mate paired FASTQ files per sample. They may be gzip compressed.

Many sequencing centers generate multiple files per mate, as from separate lanes, runs, or sequencer machines. If so, there is a R utility script called mergeMultipleFastqs.R (or similar), perhaps customized to a particular sequencing center’s file naming conventions. It can be run to merge and compress multi-part FASTQ files to convert them to the DuffyNGS expectations.

Once ready, all the raw FASTQ files are located in a folder of your choosing, and the option “fastqData.path” is set to point at that location. In this way, the Annotation.txt file only needs to specify the basenames of FASTQ files, and the Options.txt file can specify the pathname to those files.

**2.3. Set up Environment Variables**

Environment variables are typically set in your “~/.bashrc” (or similar) file, such that they get initialized every time a Linux shell environment gets started. Several are used to set up paths to needed programs and folders. Especially ones expected by the DuffyNGS pipeline, that may include:

export NGS\_PATH="/cidr/primary/morrison/NGS"

export R\_PROGRAM="/depot/install/R-3.0.2/bin/R"

export SAMTOOLS\_PATH="/cidr/home/bmorrison/bin"

export BOWTIE\_PATH="/cidr/primary/morrison/NGS/bin"

export BOWTIE\_PROGRAM="/cidr/primary/morrison/NGS/bin/bowtie"

export BOWTIE\_INDEX\_PATH="/cidr/primary/morrison/NGS/BowtieIndexes"

export BOWTIE2\_PATH="/cidr/primary/morrison/NGS/bin"

export BOWTIE2\_PROGRAM="${BOWTIE2\_PATH}/bowtie2"

export BOWTIE2\_INDEX\_PATH="/cidr/primary/morrison/NGS/Bowtie2Indexes"

export BLAST\_PATH="/cidr/home/bmorrison/Blast/bin"

export BLAST\_PROGRAM="${BLAST\_PATH}/blastn"

export BLASTINDEX\_PATH="/cidr/primary/morrison/Blast\_Indexes"

export PYTHON\_PATH="/cidr/home/bmorrison/bin"

Of those, the most important may be “R\_PROGRAM”, so you have complete control over which version of R the pipeline is run under. Note that the ‘Options.txt’ file can expand environment variables at runtime to allow symbolic folder naming.

**2.4 Try to run the “QuickQC” pipeline step:**

This is where we finally try to put all the pieces together and see if we can run the small DuffyNGS quality control pipeline step on a sample. To successfully run, all the installed software, folder paths, system and program settings, etc., need to be correctly configured. It tends to be an iterative, trial and error process involving:

A) Option value settings in the “Options.txt” file. The most important tend to be paths to FASTQ files and the pathnames and filenames of the Bowtie indexes. More details about the various options and their effects are mentioned below in Part 3.

B) Environment variables and folder/file permission issues.

C) Try to run the ‘QuickQC’ pileline on one sample:

1: Change directory to the experiment folder for this dataset family, and start up R.

2: Source the “pipelineSetup.R” script.

3: Try to QuickQC one sample: type: pipe.QuickQC( <sampleID>) where the sample ID is one character string that must match one row entry in the “Annotation.txt” file.

4: Repeatedly fix any errors until you successfully complete one QC run. A typical QC run takes roughly 10-30 minutes, depending on the organism and single vs paired reads.

Once we have completed a QuickQC run, we can be reasonably sure we are almost ready to officially process datasets for real analysis.

**Part 3: Process All Your Data**

Once you have a working setup, it’s time to fully process all the data for all the samples in this experiment family. This has a few main steps.

**3.1. Decide and set alignment parameter details:**

By inspecting the QuickQC results of a few samples, you can get a sense for the overall alignment results, and then tailor various parameters to get optimal alignment success for this experiment. A few of the most important things to check are:

* Base trimming - inspect the rate of base mismatch errors along the length of the raw reads. See files in folder “Quick.QC/AlignStats/<sampleID>/”. Images about Base mismatches and positions show how many bases are not matching the genome, and how they land within the raw read. See options ‘trim5’, ‘trim3’.
* Single or dual organism target - inspect the ‘Alignment.Pie’ image in that same folder, for a visual overview of how well raw reads aligned. Too high a fraction of ‘No Hit’ reads is a warning that the current target organism is not a good enough match to the raw data. Typical reasons include: a large fraction of some second organism DNA in the sample; foreign contamination; etc. See option ‘targetID’, and the No Hits summaries, described below.
* Strand specificity of aligned reads - inspect the read pileups of the top expressed genes. See files in folder “Quick.QC/html”. You can see read counts, a measure of strand specificity, and click to view read pileup images. These can quickly show if the raw data is strand specific and if the annotation parameter for read sense is correct or needs reversing. See annotation field ‘ReadSense’.
* ‘No Hits’ Consensus Reads summary - The Quick QC pipeline does a cursory pass at quantifying and identifying the reads that failed to align to the target organism(s). It assembles the unaligned raw reads into larger DNA ‘Consensus Read’ contigs and then uses BLAST to assign most likely organism identifiers, giving you an overview of the most plentiful constructs in your data that not align to the target organism. See files in folder “Quick.QC/CR/<sampleID>/”. Typical ‘foreign’ findings include: Illumina adapter reads, Phi-X spike-in controls, bacterial contaminants, novel regions of the target organism with excessive variation from the reference genome, or the presence of a second organism that may warrant using a dual organism target, etc.

Other parameters that affect how the raw data get ultimately treated are set in the Annotation.txt file, and include:

* ‘PairedEnd’ - are the raw files to be treated as paired end mate pair data. When the data is paired end, many of the QC evaluations are performed on each mate separately, to help determine if any issues are Mate\_1 or Mate\_2 specific.
* ‘ReadSense’ - specifies whether the raw reads represent sense or antisense strands. Mostly an issue when working with strand specific reads and wanting strand specific gene expression. Can usually only be set correctly by trial and error, as it is ultimately determined by how the original RNA was isolated.
* ‘StrandSpecific’ - affects how the aligned reads get assigned to strands. Mostly effects paired end reads when the Illumina libraries were generated with strand-specific kits. When both ‘PairedEnd” and “StrandSpecifc” are TRUE, alignment is run in full mate pair mode, and the results are strand specifically assigned. Then at the transcriptome gene expression step, only reads aligned to the coding strand are included in the read count and RPKM values. Note that the ‘Ribo Clearing’ alignment step can interfere with PairedEnd/StrandSpecific runtime behavior, since both mates may have unequal Ribo Clearing alignment outcomes. Add option “forcePairedEnd” TRUE to override default behavior and force paired end alignment mode during all alignment steps.
* ‘ExonsOnly’ - specifies how to make use of the gene/exon annotation when generating expression values for genes; determines which region boundaries are used for summing up read pileups. Genomes with high quality exon boundary annotation can benefit from turning on this parameter. Then only reads landing inside the exons count toward gene expression, and reads falling in untranslated regions (UTRs) will be ignored. But for genomes with poor exon annotation, leave this option off (default) to count all reads landing anywhere inside the gene.

**3.2. Configure batch job submission settings:**

Inside the “multiPipeline.sh” bash script are SGE/SLURM/etc arguments that handle job control and needed resources. These may need to be altered, based on the target organism, number of raw reads, the syntax of your cluster job control language, etc. You want to request enough resources so the job completes without being killed by the scheduler, but not so much that the cluster is under-loaded or you’re paying for unused resources. It’s a bit of an iterative subjective process. For ChIP-seq & RIP-seq data, once you have a good set of job control arguments for the main pipeline, copy those over to the peak picker bash script “multiPeakPick.sh” (it needs less memory than the main alignment pipeline).

**3.3. Submission of one or all samples to the cluster:**

Invoke the “multiPipeline.sh” bash script on the Unix command line. It accepts either one SampleID argument or none. If no explicit argument, it opens the Annotation.txt file and submits all samples to the cluster. Otherwise it submits just the one named sample to the cluster. Near the bottom of the script is a “sleep” step, to stagger the job submissions by a user settable amount of time (in seconds). This script calls the main top-level pipeline step on a sample, that fully processes all the raw FASTQ read data into wiggle track read pileups, transcriptome, etc. It is a batch only processing pipeline that does not use or need X11 graphics. If you are doing ChIP-seq or RIP-seq data that needs peak calling and peak curve fitting, that step will need to be done later in a separate interactive job session.

**3.4. Invoking a ‘sinteractive’ or ‘qlogin’ interactive session:**

Many of the DuffyNGS tasks require graphics, or otherwise benefit from an interactive session instead of batch job mode. There will be some cluster command that gives you a large resource session on a compute node. At CIDR, there is a command called “sinteractive” that let’s you select resources on the fly. At ISB, for the Baliga lab, a typical session is invoked with the command ‘qlogin’, as in:

qlogin -pty y -m n -q baliga -P Bal\_bmorriso -l mem\_free=20G

It is often easiest to set up a bash alias, as in:

alias qlogin=”qlogin -pty y -m n -q baliga -P Bal\_bmorriso -l mem\_free=20G”

alias sinteractive='/depot/sinteractive/0.0.2/sinteractive'

Once the qlogin session opens, ‘cd’ to the folder for this dataset family. You are now ready to either:

A) invoke various Unix scripts, such as the ChIP peak drawing scripts, from the bash command line, or more generally,

B) you can start up R, source the “pipelineSetup.R” script that loads the DuffyNGS packages, and then use the full suite of DuffyNGS tools interactively.

**3.5. Typical workflow vignette:**

Inline below is a brief example of a typical case control experiment workflow. Intended more as a outline, rather than a real-life setup.

# some DuffyNGS R commands to process example datasets

# more for information purposes than for actual processing...

# load the DuffyNGS and DuffyTools R packages,

# load the annotation table and get the vector of all sample IDs

source( "pipelineSetup.R")

sids <- annT$SampleID

# run the main alignment pipeline on all samples

# more typical is to do this step as multiple batch jobs on a cluster

for ( s in sids) pipeline( s)

# if you want to make read pileup plots for every sample, to judge

# strandedness, etc., creates files and plots in the "results/html/"

# folder

for ( s in sids) pipe.TranscriptToHTML( s, N=50, tailWidth=2500)

# gather all the main alignment metrics into one summary file, written

# to the current working directory

pipe.ExtractPipelineSummaryDetails( sids)

# the generic versatile "read pileups" gene expression plot function:

# accepts one or more sampleIDs, one or more geneIDs

# many optional arguments that affect the plot appearance

checkX11()

pipe.PlotGene( sids, "MSMEG\_0001")

pipe.PlotGene( sids, c("MSMEG\_0004","MSMEG\_0088"), plotType="l", lwd=2,

tailWidth=2500, minYmax=20, legend.cex=0.5,

plotFormat="pdf")

# turn all the transcriptome files into one matrix of gene expression

# that you can then send to cluster tools, PCA, etc.

fset <- file.path( "results/transcript/", paste( sids,

"Msmeg.Transcript.txt", sep="."))

m <- expressionFileSetToMatrix( fset, sids, verbose=T)

plot( expressionCluster( m))

Sys.sleep( 5)

matrix.PCAplot( m, col=annT$Color)

# turn all transcriptomes into the 2-way DE Ratio files needed for

# the 'RoundRobin' DE tool

runDiffExpression( sids, groupSet=annT$Group, missingOnly=TRUE)

# run the main Differential Expression (DE) pipes on all samples,

# creates 5 DE folders of results from each of the 5 DE tools,

# and then merges those into one “MetaResults” folder.

pipe.MetaResults( sids, folder="Induced.v.Uninduced",

groupColumn=”Group.Induced”,

colorColumn=”Color.Induced”)

# run the main Differential Gene Sets (Pathways) pipe on all samples

# creates 4 GeneSet folders of pathway results, and then merges those

# results into one “MetaGeneSets” folder.

pipe.MetaGeneSets( sids, folder="Induced.v.Uninduced",

groupColumn=”Group.Induced”)

# delete unneeded intermediate BAM files

cleanupBAMfiles()

# done for now

**Part 4: ChIP-seq & RIP-seq Data**

Moved down to Parts 8 & 9 below…

**Part 5: Miscellaneous Issues**

**5.1) Reprocessing data when things change:**

General rules of thumb:

* If the genomic sequence changes, then all Bowtie indexes must be rebuilt, and all samples need to be completely rerun through the entire pipeline workflow.
* If the gene annotation changes (i.e. the ‘MapSet’ that defines the organism’s layout on its genome), then the main pipeline alignment results are fine. Only the transcriptomes and any Differential Expression results need to be rerun. See functions ‘runTranscript()’ and/or ‘pipe.Transcriptome()’
* If the set of genes to be filtered out during the RiboClearing pipeline step gets updated, then the RiboClearing Bowtie indexes need to be rebuilt, and then all samples need to be completely rerun through the entire pipeline workflow.
* Turning strand specificity on or off: the effect of the strand specific settings on the pipeline happens at 2 key steps in the pipeline -- during Bowtie alignment and then during wiggle track creation from the read pileups. If strand specificity was off when the raw data was first processed, then the strand information was not included during Bowtie alignment. All samples must be fully reprocessed after turning both strand specificity and paired end mode on in the Annotation.txt file. If previously on, and we want non-strand specific transcriptomes, then turn off the strand field in the annotation and just rerun the transciptome steps with optional argument “loadWIG=TRUE”. If previously on, but you determine that the read sense was not correct, then correct the “ReadSense” in the annotation file, and just rerun the transciptome steps with optional argument “loadWIG=TRUE”.

**5.2) Further remarks on strand specific data:**

Correctly incorporating strand specific reads has several components, and the “pipe.QuickQC()” pipeline step is the best method to assure that all elements are properly configured.

* Annotation.txt fields “StrandSpecific” set to TRUE, and “ReadSense” set to the correct sense. The only sure way to know the correct read sense is to run the QuickQC tool and then inspect the gene plots in the “html” results subfolder. Confirm the displayed strand pileups match the expected coding strand, and then if needed, reverse the read sense in the Annotation.txt file and repeat the QuickQC step. To assist with this, the tool tries to auto-decide if the strand is wrong, and displays a warning message on each gene plot.
* If the samples were already run as paired end strand specific, and just the read sense is backwards, only repeat the transcriptome steps with optional argument “loadWIG=TRUE”. Otherwise, correct the read sense, turn on both paired end and strand specific, and then rerun the full pipeline on each sample.

**Part 6: Navigating the Results**

The DuffyNGS pipeline basically turns raw .FASTQ read data into .BAM files and transcriptome files. But along the way it creates a wide variety of result folders and files, and both the organization and volume of results can seem overwhelming. This section attempts to give both bioinformatics developers/programmers and the ‘end-user’ scientist consumers of the data an overview to help navigate and understand the results generated by the pipeline. As the number of result types is large, and each sample produces many files, a ‘hierarchy of subfolders’ organizational motif is employed. Each result folder and its typical contents will be described, roughly in the order the data get generated by the pipeline. Many of the comments are intended for just the bioinformatician or the scientist -- hopefully the context makes it clear.

**6.1. The ‘results’ folder and its subfolders:**

The full set of all data generated by the pipeline will be written under a user-defined folder, named by option “results.path” in the “Options.txt” file (see section 2.1.B). By default, it is called “results”. New or altered processing can be kept along side previous results by just renaming the results path, and then rerunning the pipeline. This allows comparisons between target genomes, alignment settings, annotation releases, etc. All future parts of this section refer to folders under this top level “results” folder.

**6.1.1. The ‘align’ subfolder:**

The alignment BAM files for every sample will be written here. These contain all reads successfully aligned to the target genome(s). They are binary (non-human-readable) format, that can be viewed by the SAMTOOLS VIEW utility.

**6.1.2. The ‘riboClear’ subfolder:**

If ribosomal clearing of unwanted transcripts was performed, (see option “RiboIndex”), this folder contains the alignment BAM files for reads that successfully aligned to those genes slated for clearing. The list of genes flagged for clearing is hardcoded into the annotation MapSet for the genome(s), and is not user-adjustable.

**6.1.3. The ‘splicing’ subfolder:**

If the genome(s) contain multiple exon gene models, and splice alignment was performed, (see option “SpliceIndex”), this folder contains the alignment BAM files for reads that successfully aligned to exon-exon junctions. Other files that summarize exon splice statistics will also be found here.

**6.1.4. The ‘fastq’ subfolder:**

All reads that failed to align to any of the above targets are collected as FASTQ ‘NoHit’ files here. These reads may be used by other pipeline tools that search the raw reads for non-reference alternate sequences, etc.

**6.1.5. The ‘wig’ subfolder:**

After all alignment steps, reads assigned to chromosomes and splice junctions are gathered and counted to produce read pileup wiggle track data. These WIG files are binary (non-human-readable) and contain the nucleotide resolution read depth information for both strands for all chromosomes of the genome(s).

**6.1.6. The ‘transcript’ subfolder:**

This folder contains the first principal result type wanted by the scientist researcher, files of gene expression for all genes in each genome. These transcriptome files quantify gene expression in both relative (RPKM) and absolute (raw read counts) units. Each RNA-seq or DNA-seq sample dataset will produce one transcriptome for each genome in the current target. In this way, mixed targets like “Human+Parasite” will generate 2 transcriptomes from each sample.

Transcriptome files can be converted to an orthologous species by the DuffyNGS function “orthologTransform()”, which maps genes to their orthologs (where possible) to generate a best estimate of that sample’s gene expression in a different species.

**6.1.7. The ‘summary’ subfolder:**

This folder contains small summary text files that quantify the success and/or failure of each pipeline alignment step for each sample. A single file that summarizes the alignment results and breakdown by species for a set of samples can be generated using DuffyNGS function extractPipelineSummaryDetails() and is typically written to the top-level experiment folder with a name like “PipelineSummaryDetails.csv” or similar.

**6.1.8. The ‘html’ subfolder:**

This folder may contain small HTML files that have hyperlinks to images of read pileup plots for the top expressed genes in all transcriptomes for a sample. These files are generated by DuffyNGS function “pipe.TranscriptToHTML()”, and are typically used to assess pipeline runtime annotation settings like “ReadSense” and “StrandSpecific” (see section 2.1.A), and/or the overall success of how well the raw reads aligned to the selected genome(s).

**6.1.9. The ‘ratios’ subfolder:**

This folder contains files of 2-sample gene expression differences for all genes in each genome. These ratio files quantify fold change in RPKM and assign a P-value of significance for every gene between exactly 2 samples. Separate ratio files for each genome in the current target will be created. In this way, mixed targets like “Human+Parasite” will generate 2 ratio files from each sample pair. These files are generated by the low level differential expression tool “pipe.DiffExpression()”; and these files are required by RoundRobin, one of the high level differential expression tools described below.

**6.1.10. The ‘VariantCalls’ subfolder:**

This folder contains subfolders for each sample that has been run through the variant detection (aka SNP calling) pipeline tool “pipe.VariantCalls()”. This tool implements the standard SAMTOOLS/BCFTOOLS variant detection algorithms. Each subfolder will contain a detailed .VCF.txt file of SNP calls for each chromosome, and a Summary.VCF.txt that combines, cleans, and organizes all the chromosomes into one file SNP call file. Additionally, “pipe.VariantCalls2html()” can be run subsequently to make a HTML file and subfolder of SNP plot images for the highest scoring SNPs.

There will be several other subfolders under the main “results” folder, some of which are beyond the scope of this write-up. Other folders of note contain results related to differential gene expression, discussed next.

**6.2. Differential Gene Expression (DE) and its subfolders:**

The detection of differentially expressed genes is one of the main objectives of the DuffyNGS package, and as such, there are several subfolders of results related to finding DE genes. The package makes use of 5 different DE tools and merges those 5 separate DE results to try to give one final overall DE result per categorical group comparison, with as little DE tool bias as possible.

Each tool has its own methodology, input and output details, and generates its own measures of fold change and significance for each gene, as detailed below. Then all results from all tools are merged by suitable averaging techniques to create the final DE meta results.

To help keep distinct DE comparisons separate, and to prevent file name collisions when writing new result files, each tool’s DE comparison gets stored into its own named subfolder, where the folder name contains both the organism species prefix and a user given phrase that describes the categorical groups being compared.

**6.2.1. The 5 DE tools:**

All DE tools create result files with similar names and content, differing only in their internal methods for turning gene expression input data into DE fold change and significance output data. The tools will be briefly introduced and then the result files will be briefly described.

**6.2.1.1 Round Robin:**

Akin to a round robin sports competition strategy where every team plays every team. Each sample is compared in a pairwise fashion against every other sample from other categorical groups. Final fold change and significance are the average over all 2-way compares. Uses RPKM normalized expression ratios, found in the “ratios” folder. Written by Robert Morrison, unpublished.

**6.2.1.2 Rank Product:**

A pair-wise strategy that does all possible 2-sample DE comparisons and summarizes the gene rank locations. Significance is measured as the product of all ranks. Uses RPKM normalized expression from the “transcript” folder. Written by Breitling, ***FEBS Letters*** (2004).

**6.2.1.3 SAM:**

“Significance Analysis of Microarrays”. A group-wise strategy that looks at the distribution of expression values within each group, and measures fold change and significance by the “distance” between the 2 groups. Uses RPKM normalized expression from the “transcript” folder. Written by Tusher, ***PNAS*** (2001).

**6.2.1.4 EdgeR:**

A group-wise strategy that looks at the ‘dispersion’ of read counts within each group, and measures fold change and significance by means of a negative binomial distribution. Uses un-normalized raw read counts from the “transcript” folder. Written by Robinson, ***Biostatistics*** (2008).

**6.2.1.5 DESeq & DESeq2:**

Another group-wise strategy that looks at the ‘dispersion’ of read counts within each group, and measures fold change and significance by means of a negative binomial distribution. Uses un-normalized raw read counts from the “transcript” folder. Written by Anders, **Genome Biology** (2010). (the newer DESeq2 does not correctly support groups with only a single sample, so original DESeq is used in those situations).

**6.2.2. MetaResults:**

Final DE results are created by merging result files from all 5 separate DE tools’ result files. This process helps to overcome weaknesses in each of the 5 approaches and reward consistency among the 5 tools. Intuitively, genes called significantly differentially expressed by all 5 tools are more believable. Merging is performed by low level function “metaResults()” which uses a weighted average of fold change, P-values, and rank positions from the 5 DE tools to determine the final ranking of every gene. Fold change values are averaged by arithmetic mean, P-values by geometric mean, and ranks are averaged by square-root mean (generalized mean to the ½ power).

**6.2.3. DE files:**

Each DE results folder will contain a small number of files for each comparison group. Each filename will contain several descriptive naming fragments to help precisely specify the file contents: 1) the comparison group; 2) the species prefix; 3) the DE tool abbreviation; 4) the DE directional suffix (typically one of “UP” or “DOWN”); and lastly 5) the file type suffix, as in HTML, CSV, TXT, etc. Depending on the DE tool, there will be 2-6 separate files for each comparison group.

In the general case, each comparison group is being evaluated against all other comparison groups at the exact same time, in what can be thought of as a K-way comparison. In the trivial case of a 2-group comparison, there will be symmetric results (e.g. “UP in group A” is roughly identical to “DOWN in Group B”). In all cases, the name of the subfolder containing the DE files is intended to convey exactly which groups and comparison are in that subfolder.

For each comparison group, there will be files:

A) “UP.txt”, “DOWN.txt” and/or “Ratio.txt”: these text files contain all the genes in the genome, sorted by up or down regulation. Each file contains all columns of DE details as generated by the specific DE tool. These file are tab delimited, openable in Excel, etc.; and intended as a complete result of the comparison question being asked.

B) “UP.html”, “DOWN.html”: these hypertext files contain just the top N most DE genes (N usually 100-200 or so), with hyperlinks to some form of gene expression plot images. These files are openable with any web browser, and intended as an easy way to view just the most interesting subset of a comparison question.

C) “UP/DOWN.GeneCellTypeEnrichment.csv”: these files (MetaResults only) are supplemental results that convey information about cell type enrichment (or parasite life cycle in the case of parasite genomes). Taking the genes called significantly DE, and looking at their associated cell types, these files suggest if the comparison group as a whole is over or under enriched for any specific cell types.

D) “pngPlots”: a folder of gene expression plots, for all the genes in any of the HTML files.

**6.2.4. Commands to invoke Differential Expression functions:**

Each of the 5 DE tools and the Meta DE tool have almost identical command line arguments, described in more detail below. General format of all the commands, and their default settings, is:

pipe.MetaResults( sampleIDset, folderName, groupColumn=”Group”,

colorColumn=”Color”, speciesID=getCurrentSpecies(),

results.path=NULL, annotationFile=”Annotation.txt”,

optionsFile=”Options.txt”,doDE=TRUE, PLOT.FUN=NULL,

… )

Of all the command line arguments, only the set of sample IDs to compare and the name for the results subfolder are required. All others have default values. Any of the 5 DE tools can be called directly via “pipe.RoundRobin()”, “pipe.RankProduct()”, “pipe.SAM()”, “pipe.EdgeR()”, or “pipe.DESeq()”. But the default behavior of “pipe.MetaResults()” is to call all 5 DE tools in parallel and then to combine the 5 sets of results into a final consensus answer. The main arguments are:

**sampleIDset** -- a vector of 2 or more sample IDs to be compared.

**folderName** -- a character string that very briefly captures the essence of this one comparison being performed. It gets used to name the subfolder that will contain all the DE results generated for this comparison. In a typical experiment there may be many comparisons of interest, so naming each folder carefully is very helpful to minimize future confusion. This folder name will be prefixed with the current species, to allow keeping mixed organism results cleanly separate.

**groupColumn (default=”Group”)** -- the name of one column in the annotation file that defines the groups being compared.

**colorColumn (default=”Color”)** -- the name of one column in the annotation file that defines the display color for each group, used by various image generating functions.

**results.path (default=NULL)** -- the name of the top level results folder that contains all data to be used (transcriptomes, etc.) and all result files that will be created. The default value means to use the value of the “results.path” option in the “Options.txt” file.

**doDE (default=TRUE)** -- should all the low level differential expression calculations be performed?. When FALSE, just replot those genes already found to be DE from a previous call to the tool.

**PLOT.FUN (default=NULL)** -- the name of an R function to call for generating images of gene expression. It must be a function that accepts a single value that is a gene ID. Default behavior is to call the tool that generates raw DNA read pileups in the region of the gene. Note that this default tool can be very slow for large numbers of samples. “NA” means to not call any plotting function at all.

**… (default=none)** -- other arguments, passed down to lower level functions.

**6.3. Differential Pathways (aka ‘Gene Sets’) and their subfolders:**

The detection of differentially expressed pathways is the second main objective of the DuffyNGS package, and as such, there are several subfolders of pathway results related to any one DE gene analysis. The package makes use of 4 different differential pathway tools and merges those 4 separate results to try to give one final overall differential pathway result per categorical group comparison, with as little tool bias as possible.

As a note on terminology, we use the terms ‘Pathway’, ‘Module’, and ‘GeneSet’ interchangeably. They all simply denote predefined sets of genes to be treated as single groups.

**6.3.1. Pathway Results subfolders & files:**

Located under the MetaResults folder for any one DE comparison, there may be 1 to 4 additional subfolders that contain Pathway/Module/GeneSet comparison results. These all seek to provide a higher level analysis than looking just at individual genes, by looking at related collections of genes to detect differential expression of entire sets of gene, Each tool starts from the DE results in the current folder and evaluates a large set of pre-defined gene sets from various public sources, to find those with the best overlap and significance among the genes called DE in this comparison. Each tool and its subfolder are very briefly introduced:

**6.3.1.1 CombinedGeneSets:** uses all the differential expression results, and looks at how sets of genes behave as a collection of mass points between the different comparison groups. When most all the genes in a network/pathway are strongly up/down regulated, their ‘mass’ shifts relative to the other comparison groups. We can put a P-value on how unlikely such a distribution shift is. As the calculations are implemented via density functions on each group, this analysis is sometimes referred to as Gene Set Density analysis.

**6.3.1.2 RadarPlots:** starts from the matrix of gene expression for all samples (instead of DE), transforms the data into M-values, and then consolidates the genes and samples down to reflect the average behavior of the group relative to the other comparison groups. These images can be very effective in talks as they seem visually intuitive.

**6.3.1.3 QuSage:** starts from the matrix of gene expression for all samples (instead of DE), transforms the data into M-values, and then compares confidence intervals to reflect the average behavior of each group relative to the other comparison groups. See: G.Yaari,

Nucleic Acids Res. 2013

**6.3.1.4 Enrichment:** starts from the DE results, and looks at subset overlap (think “Venn Diagrams”), to find groups of genes in the top up-regulated set that are larger than expected by chance.

**6.3.1.5 MetaGeneSets:** combines all 4 GeneSet analyses tool results, by appropriate averaging methods, to give a final overall ranking of the top up and down regulated gene sets. As each of the 4 gene set tools has their strengths and weaknesses, this averaging works to find the best true DE gene sets, and discounts false positives reported by only a subset of tools.

**6.3.2. Commands to invoke Differential Pathway functions:**

Each of the 4 GeneSet tools and the Meta GeneSet tool have almost identical command line arguments, described in more detail below. General format of all the commands, and their default settings, is:

pipe.MetaGeneSets( sampleIDset, folderName, groupColumn=”Group”,

colorColumn=”Color”, speciesID=getCurrentSpecies(),

results.path=NULL, annotationFile=”Annotation.txt”,

optionsFile=”Options.txt”,doGeneSets=TRUE, doMissing=TRUE, baselineGroup=NULL, label=””,

NgeneSets=500)

Of all the command line arguments, only the set of sample IDs to compare and the name for the results subfolder are required. All others have default values. The 4 GeneSet tools can be called directly via “pipe.GeneSetDensity()”, “pipe.RadarPlots()”, “pipe.QuSage()”, and “pipe.GenSetEnrichment()”. The default behavior of “pipe.MetaGeneSets()” is to only use those GeneSet results that already exist. Alternatively, we can force the calls to all 4 GeneSet tools with argument “do.missing=TRUE”. Lastly, the function then combine the 4 sets of results into a final consensus answer. The main arguments are:

**sampleIDset** -- a vector of 2 or more sample IDs to be compared.

**folderName** -- a character string that very briefly captures the essence of this one comparison being performed. This tells the GeneSet tools which folder of previously generated DE gene results to use as the input data for finding differential gene sets.

**groupColumn (default=”Group”)** -- the name of one column in the annotation file that defines the groups being compared.

**colorColumn (default=”Color”)** -- the name of one column in the annotation file that defines the display color for each group, used by various image generating functions.

**results.path (default=NULL)** -- the name of the top level results folder that contains all data to be used (transcriptomes, etc.) and all result files that will be created. The default value means to use the value of the “results.path” option in the “Options.txt” file.

**doMissing (default=TRUE)** -- should all the low level differential gene set calculations be performed? When FALSE, just use those gene set results already found from previous calls to the low level tools defined above.

**doGeneSets(default=TRUE)** -- should all the low level differential gene set calculations be re-calculated? When FALSE, just use those gene set results already found from previous calls to the low level tools defined above.

**6.4. Plotting Images of Gene Expression:**

The ability to visualize the read pileups and/or RPKM values of expressed genes is essential to fully understanding the data. Often a visual image of the read pileups can convey far more detailed information than tabular lists of gene expression ever could. To that end, the package has a variety of low level plotting routines. For simplicity, there is one gene plot function that tries to handle a wide variety of situations and dispatch to the best low level function.

Use function “pipe.PlotGene()”, with the following main arguments:

**sampleIDs** — a vector of 1 or more sample IDs.

**genes** — a vector of 1 or more GeneID.

**colorColumn (default = "Color")** — the name of a column in the annotation file to give the color used for each sample (only used when 2+ samples given).

**PLOT.FUN (default = NULL)** — an optional function for plotting, to allow complete customization of how a gene plot gets created.

**tailWidth (default = 1000)** — number of bases outside the gene boundary to include in the image. Allows control of how much genomic context is shown.

**plotType (default = “boxes”)** — controls how the pileups are presented. Default is filled colored rectangles. Other choices include: “lines”, “segments”, “addedLines”, “none”.

**useLog (default = FALSE)** — present the Y axis as either linear or log scale.

**addStrands** (default =FALSE for 1 sample, TRUE for 2+ samples) — controls how the forward and reverse strand read depths are displayed w.r.t. Y axis.

**forceYmax ( default = NULL)** — overrides the default Y axis scaling. By default we auto-scale so the named gene fills the Y axis. Specify a numeric value to override auto-scaling.

**minYmax (default = 10)** — sets a minimum Y axis scaling. Useful for generating multiple gene plots with identical Y axis limits.

forceYmax = NULL, minYmax = 10, showDetectability = TRUE — controls the drawing of the detectability feature along the lower X axis.

**…** — any other optional arguments passed to lower level plot functions.

There are a few other arguments for forcing creation of .PNG or .PDF files automatically, but these are in flux at present. Also, the size and aspect ratio of the current graphics window affects the gene plot images. There are many occasions where resizing the plot window and then recalling the same plot function will improve the overall appearance of the image.

Additionally, there are cases where a stand alone R function can better meet particular needs. One example is given in the R source file called “geneExpressionPlots.R” which contains a highly customizable method for displaying many samples across multiple comparison groups all together in a single image.

**Part 7: Variant Detection (SNPs & Deletions)**

While typically done only on whole genome sequencing (WGS) data, all SNP and variant detection tools operate on all types of aligned read data. The principal methods are described briefly below.

**7.1. Detection of large scale deletions and copy number differences:**

Running WGS data through the standard RNA-seq Transcriptome and Differential Expression pipelines is an easy way to detect deletions and copy number differences. Deletions will appear as strongly down-regulated genes, and CNV will appear as up-regulated genes. Large deletions will be signaled by many adjacently located genes all being down-regulated together.

Another feature that seems to appear often in bacterial samples, that can look like a CNV event, is when foreign plasmids, phages, or other contaminants are present in the sample. They will typically pile up like extremely deep read depths in certain localized genes, much deeper than the surrounding mean chromosomal read depth.

There is one function specifically intended to detect deletions in annotated genes, called “pipe.FindDeletions()”. It assesses how many base locations inside the gene boundary have less read depth than a given threshold (default = 1), and quantifies each gene as being some ‘percentage\_deleted’, by dividing that count by the length of the gene. It returns a data.frame where the most deleted genes are at the top. Note that very low coverage WGS data can give a false sense of true deletions, and the RNA-seq data will call non-expressed genes as being deleted.

**7.2. SNP calling:**

Running either WGS data or RNA-seq data through the SNP calling pipeline is equally meaningful, with a few caveats. RNA-seq reads that span exon boundaries will often align with what appear to be SNPs very near the exon edge, most often just into the intron. These are artifacts that should be ignored. WGS reads will never give this type of false artifact. Most SNP calling algorithms use the read depth in the vicinity of a SNP site as part of their statistical measure. WGS data is typically of more uniform depth, and more like what the SNP calling algorithm expects. The highly variant read depth of RNA-seq may falsely confuse the SNP algorithm, but there is no meaningful remedy. The best work-around is to give the tool a very lax P-value threshold, so more SNP candidate sites get flagged as probable SNPs and then use more rigorous down-stream selection criteria to find what is real.

In bacterial samples that happen to have plasmid or phage contaminants, the CNV-looking pileups mentioned above will almost always present with a flurry of SNPs on the flanks of the unexpected pileup, where phage/plasmid sequence and the bacterial genome sequence diverge. These SNP jackpots are almost always accompanied by a sharp increase in read depth in that local area.

The basics of the SNP calling pipeline are as follows:

**7.2.1. Call SNPs:**

Run function “pipe.VariantCalls()” on each sample. This routine evaluates all nocleotide locations on all chromosomes, and write files of detected SNP details to a subfolder under /results/VariantCalls/. Main arguments include:

**prop.variant (default = 0.5)** – keep all SNPs with a P-value below this level. Higher values keep more partial SNPs.

**min.depth (default = 5)** – only evaluate locations with enough reads to be trusted. Very low read depth can let random sequencing errors look like SNPs. We like to have enough reads to give a strong consensus that the allele does not match the reference.

**7.2.2. Generate simple SNP image plots for a single sample:**

Run function “pipe.VariantCalls2html()” on a sample. This routine creates one HTML file with hyperlinks to images for each of the best high scoring SNPs detected in the sample. Main arguments include:

**max.plots (default = 200)** – maximum number of SNPs to plot.

**min.score (default = 100)** – which SNPs to plot, ignore lower quality sites.

**altAA.only (default = TRUE)** – limit plotting to SNPs that change the gene’s protein coding sequence.

**tailWidth (default = 46)** – number of extra nucleotides on either side of the SNP site to include in the plot image. Larger numbers show more context, but can make the image hard to interpret. Smaller numbers show less context, but give a cleaner image.

**7.2.3. Find differential SNPs between experimental groups:**

Run function “pipe.VariantComparison()” on a set of samples from 2 or more groups/conditions. This routine evaluates all SNP calls in the given samples, and attempts to flag sites with the largest differences.

Overall, the tool performs quite poorly, being easily misled by low quality partial SNPs as reported by the SAMTOOLS/BCFTOOLS workflow.. Still, it’s better than nothing… The function returns a data.frame of the best SNP differences, that can be passed to a plotting function (see 2.4 below). The tool performs best given exactly 2 groups, where the SNP patterns between the groups are somewhat conserved, but can handle more groups with high SNP variability.

Main arguments include:

**sampleIDset** – the set of 2 or more samples to be compared.

**groupSet** - the group names for each sample, used to partition the samples into 2 or more groups.

**min.deltaScore (default = 40)** – only keep SNP sites where the difference in score between 2 groups is at least this large.

**exonOnly (default = FALSE)** – if TRUE, only consider SNPs in coding exons.

**snpOnly (default = FALSE)** – if TRUE, ignore Insertion and deletion sites. These are common in highly variant gene families, usually false SNPs, and can easily dominate the results file. Ignoring these tends to help find the most meaningful differential SNP sites.

**AAsnpOnly (default = exonOnly)** – if TRUE, only consider SNP sites where at least one group changes the gene’s protein coding sequence.

**capScore (default =100)** – if 2 or more groups have different non-reference allele SNPs at the exact same site, the difference in their scores will be set to larger than the highest possible score. This feature lets us find highly variant sites where just being ‘different from the reference’ is not the only criteria. To enable this highly variant search, raise the value above 100 up to a maximum of 200.

**7.2.4. Write & plot differential SNPs results:**

Run function “pipe.VariantCompare2html()” on the data.frame returned from the previous function described above. This routine writes a HTML file with hyperlinks to plots for each SNP site, showing the SNP and its adjacent nucleotides in all the samples in the comparison. The order of the samples in the final plots, and their relative layout, can be controlled. Writes its files and plot images to a subfolder under /results/VariantCalls/ with a folder name given explicitly by the user to describe the comparison that was performed. Main arguments include:

**tbl** – the data frame result from “pipe.VariantComparison()”

**outfile (default = "VariantCompare.html")** – the name for the subfolder and the HTML file. Make sure to use a unique name that conveys what comparison and/or samples were involved.

**sampleIDset** – the sample names in the order to be plotted.

**groupSet** – the group name for each sample, also in the order to be plotted.

**Ngenes (default = 200)** – the maximum number of SNP sites to be plotted. This plotting function can be quite slow, and most differential SNP sites are false positives, so limiting to the best subset can be useful.

**tailWidth (default =26)** – the number of adjacent nucleotides to show for each sample. Lower this value as the number of samples gets larger.

**dropNonSNPs (default = TRUE)** – remove and don’t draw sites that are not truly showing 2 different alleles. Yet another type of false positive that we try to catch and ignore.

**Part 8: ChIP-seq Data**

These types of experimental data require extra non-standard algorithms to extract meaningful results. In both cases, we begin with a genomic alignment pipeline as for WGS data. And then we follow up with highly specialized ChIP or RIP peak search and curve fitting methods. Parts of this workflow include interactive steps that needs X11 graphics. To that end, we will usually want to use an interactive session on the cluster, to get enough memory and compute resources.

**8.1. Peak Picking for ChIP-seq:**

The calling of ChIP peaks can be run in batch mode or interactively. If interactive, the tools generate a suite of peak calling and ROC curves to help inform the selection of a cutoff threshold between true peaks and noise. If batch mode is used, it is easier to guarantee that all samples get processed with identical scaling and negative control peak data. See bash scripts “peakpick.sh” and “multiPeakPick.sh”.

**8.2. Peak Drawing and Summary Files:**

The last pipeline step must be run as an interactive session. It will generate final peak images and summary tables and .hmtl results. This pipeline step is encapsulated into a pair of bash shell scripts: “peakdraw.sh” and “multiPeakDraw.sh”. They contain the final settings regarding negative control peaks and scaling. They are similar to the batch job submission scripts, but must be run from a ‘qlogin’ session.

**8.3. Details of Peak Pick Algorithm for ChIP-seq:**

Starting data for peak picking is the wiggle tracks, giving the depth of aligned reads at every nucleotide for each strand. The tool for ChIP peak picking is R function pipe.ChIPpeaks(). The global median depth for each strand is used to set the threshold for peak search. The default is 3 times median (parameter “cutoff.medians”=3). Only nucleotides above that threshold depth are considered as possible peak locations. Since ChIP peaks are read pileups, peak heights are in units of read depth, peak breadths are in units of nucleotides, and the expected breadth is dependent on the length of the raw sequencing reads. Peak breadths are defined using a “width” term based on the “half width at half height (HWHH)” paradigm, so the total footprint of a peak is roughly 6 times (+/- 3) as wide as its “width”. The search algorithm uses a rough suggestion of peak width, based on the length of raw reads, to guide detection (parameter “canonical.width”=50).

The forward strand, reverse strand, and the ‘Combined strand’ (forward + reverse), are each independently searched for peaks, using the following 5-step strategy:

**8.3.1 Local Extrema Detection:** find all nucleotides that are local maxima. The result is a list of nucleotide locations. These locations become the candidates for peak fitting. They are sorted by decreasing height, such that the tallest extrema will be fitted first.

**8.3.2. Peak Fitting:** subject the region around each given extrema point to model peak fitting, using a family of pre-defined peak shapes (gaussian, gumbel, lorentzian), via nonlinear least squares optimization. The result is a single model peak that best approximates the raw read depths over that region for that extrema point. The model peak will have several attributes defining its location, shape, and quality:

Type: the shape of the best fitting distribution, as a text string. Will be one of: ‘gaussian’, ‘gumbel’, ‘lorentz’

Center: the peak’s central location (units=nucleotides)

Width: the peak’s width (units=nucleotides). Note that the underlying detail of how “width” is interpreted depends on the distribution type. For a gaussian, ‘width’ is 1 standard deviation.

Floor: the bottom of the peak (units=reads), giving the non-specific ambient depth of reads in the flanking region outside the peak’s extent.

Height: the height of the fitted peak (units=reads), relative to its floor. This gives the height of reads inside the peak after subtracting the ambient read depth of the region. Total depth of reads at the center of the peak is thus “height + floor”.

Start,Stop: extent boundaries of the peak’s footprint (units=nucleotides)

Volume: total volume inside the model peak; similar to ‘area under the curve’, but ignoring the region of non-specific ambient reads below its floor.

Residual: the residual error of the model (units=reads), gives the volume of raw reads that are not accounted for by the best fit model.

Drift: the distance from the starting extrema location to the best fit center returned by the model (units=nucleotides)

Status: description of the result, gives various reasons why the fitted peak may be rejected

Times: a traditional measure of peak height, stating how many times higher than the global median is the total height of this fitted peak

e.g. (height+floor)/median (no units)

Score: the score for the fitted peak; an arbitrary value from the range zero to one, that measures how well the peak’s attributes {height, width, drift} resemble a ChIP peak. 1 = ideal, 0 = terrible

Pvalue: the probability that an extrema point could have this high a score, but not be a true peak. See ‘Statistical Significance’, below.

**8.3.3. Statistical Significance:** Two separate methods are used to distinguish true peaks from noise. First, to simulate the distribution of peak scores in the population of all possible peaks, a random subset of local extrema from step 1 are selected (default N=2000), fitted, and scored. Using the 95th percentile of these ‘non-peak’ scores gives one estimate of the minimum score of a true peak. Secondly, an ROC analysis is performed, using the top 100 best peaks as true positives, 1000 of the random peaks as true negatives, to yield an optimal score threshold for distinguishing true peaks from false positives. Tests using the 6 trial datasets show very good agreement between the 95th % and ROC. Each peak’s significance is then recorded in two manners:

1. The “status” string is be set to “failScore” if its peak score is below the ROC cutoff.

2. The “pvalue” is calculated as the percentage of random peaks that had a score at least as high as that peak’s. P values below 0.05 are indicative of true peaks.

**8.3.4. Results Files:** For each strand, a text file of all found peaks is written, named like “XXXXX.ChIPpeaks.YYY.STRAND.txt” suitable for use with Excel (where XXXXX is the sample name, YYY is the chromosome name, and STRAND is one of ‘Plus’, ‘Minus’, ‘Combo’). The peaks are ordered by decreasing height, so the tallest peaks are at the top of the file. Also, a ROC plot is made for each strand, showing the score cutoff, with a similar file name ending in “ROC.png”. A peak plots subfolder called “ChIP.plots” contains a collection of peak fitting images that visualize the success or failure of the peak fitting algorithm. They have names of

“rawPeak.STRAND.XX.png” where XX is the ordinal of when that peak was fit (peak ‘25’ being the 25th tallest extrema to be submitted to the peak fitter).

**8.3.5. Merge 3 Strands:**  The peak results from all 3 strands (forward, reverse, combination) are then merged, by finding triplets that align in the pattern expected of a ChIP read pileup event. Briefly, all 3 peak lists are sorted by height, and then each combo peak is visited. Given the parent’s center and width, each child peak list is searched for the tallest peak that lands within the proper side of the parent’s footprint. If one forward and one reverse child peak are found, then the final called peak’s details are taken from those 2 children. If only one child peak is found, the second child is set to a surrogate “non-peak” with terrible P value and undefined peak shape metrics. For parent peaks with no found child peaks, two surrogate “non-peak” children are assigned. The p value of each parent peak is the worst P value of its 2 children. The final result is sorted by the P value of the parent combo peaks.

The final merged result is written as both a text file and an HTML file, with a name of the form “XXXXXX.ChIPpeaks.txt”. Peaks are ordered by P value, with the best peaks at the top of the file.

All relevant columns from the 3 strands are preserved, along with a few extra columns that quantify the overall metrics about each ChIP peak. Peak fit attributes from the 3 strands get a 1-letter prefix to distinguish them (‘P’-plus strand, ‘M’-minus strand, ‘C’-combination). The additional columns include:

VPM: peak Volume (Per Million reads). This is the sum of the volumes of the forward and reverse strand fitted peaks, after normalizing for total aligned reads in the sample. This is the integral of peak-specific reads from the 2 strands without any of the ambient non-specific background reads

P\_Value: overall P value for the peak. This is the worst P value of the 2 child peaks from the separate strands

Model\_Error: percentage of total residual error in the fitted model. This is the sum of the 2 child peak residuals divided by the sum of their volumes. A lower model error percentage denotes a superior peak fit to the raw data.

DNA: the DNA subsequence that lies under the peak. The region from the center of the forward peak to the center of the reverse strand peak is extracted from the genomic DNA sequence (forward strand). If shorter than 100bp, it is extended to 100; if longer than 200bp, it is truncated – in both these cases the subsequence is centered on the midpoint between the two child peaks.

The final merged result is also written as an Excel-readable file, with a name of the form “XXXXXX.ChIPpeaks.Excel.csv”. This includes an extra column of hyperlinks to the peak plots, and re-formats the DNA column into FASTA format with GeneID descriptors for easy export to other tools. (Manual stripping of Excel-generated double quote characters may be required in some cases)

**Part 9: RIP-seq Data**

This type of experimental data is like a mashup of ChIP-seq and RNA-seq data, and requires extra non-standard algorithms to locate the regions of interest, followed by modified RNA-seq methods for finding differentially abundant RIP-seq loci.

**9.1. Details of Peak Pick Algorithm for RIP-seq:**

RIP-seq data is like a cross between RNA-seq and ChIP-seq, where the expect peaks have a very different type of expected shapes. Therefore, a separate peak picking algorithm tuned fore RIP-seq data is required. Starting data for peak picking is the wiggle tracks, giving the depth of aligned reads at every nucleotide for each strand. The tool for RIP peak picking is R function pipe.RIPpeaks(). The global median depth for each strand is used to set the threshold for peak search. The default is 3 times median (parameter “cutoff.medians”=3). Only nucleotides above that threshold depth are considered as possible peak locations. Since RIP peaks are read pileups, peak heights are in units of read depth, peak breadths are in units of nucleotides, and the expected breadth is dependent on the length of the raw sequencing reads. Peak breadths are defined using a “width” term based on the “half width at half height (HWHH)” paradigm, so the total footprint of a peak is roughly 6 times (+/- 3) as wide as its “width”. The search algorithm allows a wide range of suggested of peak widths, based on the length of raw reads and the expected size of pulled down transcripts, to guide detection (parameters “min.width”=50, “max.width=1000).

The forward strand and the reverse strand are each independently searched for peaks, using the following 5-step strategy:

**9.1.1 Local Extrema Detection:** find all nucleotides that are local maxima. The result is a list of nucleotide locations. These locations become the candidates for peak fitting. They are sorted by decreasing height, such that the tallest extrema will be fitted first.

**9.1.2. Peak Fitting:** subject the region around each given extrema point to model peak fitting, using a family of pre-defined peak shapes (gaussian, gumbel, pulse), via nonlinear least squares optimization. The result is a single model peak that best approximates the raw read depths over that region for that extrema point. The model peak will have several attributes defining its location, shape, and quality:

Type: the shape of the best fitting distribution, as a text string. Will be one of: ‘gaussian’, ‘gumbel’, ‘pulse’

Center: the peak’s central location (units=nucleotides)

Width: the peak’s width (units=nucleotides). Note that the underlying detail of how “width” is interpreted depends on the distribution type. For a gaussian, ‘width’ is 1 standard deviation.

Floor: the bottom of the peak (units=reads), giving the non-specific ambient depth of reads in the flanking region outside the peak’s extent.

Height: the height of the fitted peak (units=reads), relative to its floor. This gives the height of reads inside the peak after subtracting the ambient read depth of the region. Total depth of reads at the center of the peak is thus “height + floor”.

Start,Stop: extent boundaries of the peak’s footprint (units=nucleotides)

Volume: total volume inside the model peak; similar to ‘area under the curve’, but ignoring the region of non-specific ambient reads below its floor.

Residual: the residual error of the model (units=reads), gives the volume of raw reads that are not accounted for by the best fit model.

Drift: the distance from the starting extrema location to the best fit center returned by the model (units=nucleotides)

Status: description of the result, gives various reasons why the fitted peak may be rejected

Times: a traditional measure of peak height, stating how many times higher than the global median is the total height of this fitted peak

e.g. (height+floor)/median (no units)

Score: the score for the fitted peak; an arbitrary value from the range zero to one, that measures how well the peak’s attributes {height, width, drift} resemble a RIP peak. 1 = ideal, 0 = terrible

Pvalue: the probability that an extrema point could have this high a score, but not be a true peak. See ‘Statistical Significance’, below.

**9.1.3. Statistical Significance:** Two separate methods are used to distinguish true peaks from noise. First, to simulate the distribution of peak scores in the population of all possible peaks, a random subset of local extrema from step 1 are selected (default N=2000), fitted, and scored. Using the 95th percentile of these ‘non-peak’ scores gives one estimate of the minimum score of a true peak. Secondly, an ROC analysis is performed, using the top 100 best peaks as true positives, 1000 of the random peaks as true negatives, to yield an optimal score threshold for distinguishing true peaks from false positives. Each peak’s significance is then recorded in two manners:

1. The “status” string is be set to “failScore” if its peak score is below the ROC cutoff.

2. The “pvalue” is calculated as the percentage of random peaks that had a score at least as high as that peak’s. P values below 0.05 are indicative of true peaks.

**9.1.4. Results Files:** For each strand, a text file of all found peaks is written, named like “XXXXX.RIPpeaks.YYY.STRAND.txt” suitable for use with Excel (where XXXXX is the sample name, YYY is the chromosome name, and STRAND is one of ‘Plus’, ‘Minus’). The peaks are ordered by decreasing height, so the tallest peaks are at the top of the file. Also, a ROC plot is made for each strand, showing the score cutoff, with a similar file name ending in “ROC.png”. A peak plots subfolder called “RIP.plots” contains a collection of peak fitting images that visualize the success or failure of the peak fitting algorithm. They have names of

“rawPeak.STRAND.XX.png” where XX is the ordinal of when that peak was fit (peak ‘25’ being the 25th tallest extrema to be submitted to the peak fitter).

**9.1.5. Merge 2 Strands:**  The peak results from both strands (forward, reverse) are then merged. As there is no special paired strand relationships in RIP-seq data, just a simple join of the 2 peak lists is used.

The final merged result is written as both a text file and an HTML file, with a name of the form “XXXXXX.RIPpeaks.txt”. Peaks are ordered by P value, with the best peaks at the top of the file.

All relevant columns are preserved, along with a few extra columns that quantify the overall metrics about each RIP peak. The additional columns include:

VPM: peak Volume (Per Million reads). This is the sum of the volume of the forward or reverse strand fitted peak, after normalizing for total aligned reads in the sample. This is the integral of peak-specific reads from the coding strand without any of the ambient non-specific background reads

Model\_Error: percentage of total residual error in the fitted model. A lower model error percentage denotes a superior peak fit to the raw data.

DNA: the DNA subsequence that lies under the peak. The region from the center of the forward peak to the center of the reverse strand peak is extracted from the genomic DNA sequence (forward strand). If shorter than 100bp, it is extended to 100; if longer than 200bp, it is truncated – in both these cases the subsequence is centered on the midpoint between the two child peaks.

The final merged result is also written as an Excel-readable file, with a name of the form “XXXXXX.RIPpeaks.Excel.csv”. This includes an extra column of hyperlinks to the peak plots, and re-formats the DNA column into FASTA format with GeneID descriptors for easy export to other tools. (Manual stripping of Excel-generated double quote characters may be required in some cases)

**9.2. RIP-seq differential expression:**

Once all experimental datasets have been RIP-seq peak picked, we will use the union of all peaks to define the set of gene loci to assess for transcriptomic and differential expression analyses. We will leverage a built-in DuffyNGS feature that allows use of non-standard annotation “alternate gene maps”. These are an optional explicit table of gene names and locations that supersede the default pre-defined gene annotation for a species.

**9.2.1 Create an Alternate Gene Map:** run R function pipe.RIPpeaksToAltGeneMap() giving it the set of SampleIDs to be compared. All RIP peak results for each sample are combined, while accounting for possible overlaps and duplicates. The result is a data.frame that matches the expected layout and contents of any gene map.

**9.2.2 Make ‘Alternate Gene Map’ Transcriptomes and Ratio files:** run R function pipe.Transcriptome() on all SampleIDs to be compared. Specify optional function argument “altGeneMap” to be the data frame from 9.2.1 above, and specify using optional function argument “altGeneMapLabel” the alternate name to append to all created files. This creates a new folder under the “results/transcript/” folder that contains a complete set of alternate transcriptomes that contain expression details for all of the RIP-seq peak loci. After that, supply the same optional arguments to the pipe.DiffExpression() function, to create a subfolder of 2-way sample DE comparison files under the “results/ratios/” folder.

**9.2.3 Make ‘Alternate Gene Map’ Differential Expression Results:** now you are ready to run R function pipe.MetaResults() on all SampleIDs to be compared. Specify the exact same optional function arguments for “altGeneMap” and “altGeneMapLabel”. Be sure to set the argument “folder” to a value that has not already been used – a string like “RIPpeaks” is good. The results will be the standard type of differential expression meta results, after spawning all 5 DE tools and merging their 5 separate DE results into a final folder of differential expression results. But the features being compared on not normal genes, but instead exactly those loci that were defined as RIP-seq peaks in one or more samples. For best plot images of the results, use optional function argument “plotType”=”lines”, so you can clearly see the read pileup depths in all samples at one time.

As a side note, Pathway Analysis is not officially supported for alternate gene map DE results; but it should work just fine…