Differential translation analysis and GO term enrichment using RiboSeq and RNASeq data.

Pre-requisites

The following programs are required:

- STAR (https://github.com/alexdobin/STAR)
- samtools (http://samtools.sourceforge.net)
- HTSeq (https://htseq.readthedocs.io/)
- GNU parallel (https://www.gnu.org/software/parallel/)

The following R packages are required:

- Xtail (see below).
- Devtools (see below).
- DESeq2 (https://bioconductor.org/packages/DESeq2)
- topGO (https://bioconductor.org/packages/topGO)

Installing Xtail using Devtools

We can install Xtail directly from the Xtail GitHub, by using the Devtools package. First, however, we must install the libxml and libcurl packages.

On Ubuntu:

```
sudo apt-get install libcurl4-openssl-dev libssl-dev libxml2-dev
```

On Fedora:

```
sudo dnf install libcurl-devel libxml-devel libxml2-devel
```

Once these are installed, open an R session and run the following commands:

```
install.packages("devtools")
library("devtools")
install_github("xryanglab/xtail")
```

Re-mapping reads to the host genome

Now that Xtail is installed, we re-map the non-vRNA reads to the host genome using STAR. This process is similar to the mapping step we performed as part of the main RiboSeq pipeline; however, in this case we will report *all* multi-mapping read alignments, instead of randomly selecting a single alignment. Multi-mapping reads will then be excluded from the read counting step (see later).

Create a working directory:

```
mkdir differential_expression && cd differential_expression
```

Now let's map the non-vRNA reads to the host genome, reporting all alignments for multimapping reads:

```
#specify the path to my STAR database directory
stardbdir="/mnt/sdb1/RibosomeProfiling/STAR_DATABASES"

#specify the path to the locations of the nonvRNA.fq files
readspath="/mnt/sdf/MuLV"
```

```
for line in $(awk '{printf "%s:%s\n", $1,$3}' libraries.txt)
do
    library=$(echo $line | awk -F: '{print $1}')
    hostname=$(echo $line | awk -F: '{print $2}')
    STAR --runMode alignReads \
    --runThreadN 8 --outFileNamePrefix $library. \
    --outSAMtype BAM SortedByCoordinate \
    --outBAMsortingThreadN 8 \
    --outFilterMismatchNmax 2  \
    --outFilterIntronMotifs RemoveNoncanonicalUnannotated \
    --outMultimapperOrder Random \
    --genomeLoad LoadAndKeep \
    --limitBAMsortRAM 60000000000
    --readFilesIn $readspath/$library.nonvRNA.fq \
    --genomeDir $stardbdir/$hostname
done
```

Counting reads per gene with HTSeq

Next, we use the BAM files that STAR has output to count the numbers of reads mapping to each annotated gene in the host genome.

For a standard RiboSeq analysis, we will only count reads mapping to protein-coding genes. Therefore, we filter the GTF file that was used for creating the STAR genome index to extract only the protein-coding genes (for this example, I am working with *Rattus norvegicus* cells):

```
grep "protein_coding"
$stardbdir/Rattus_norvegicus/Rattus_norvegicus.Rnor_6.0.88.gtf >
Rattus_norvegicus.protein_coding.gtf
```

Now, we can use the python module HTSeq to count the number of reads mapped to each protein-coding gene. HTSeq will automatically exclude multi-mapping reads from individual gene counts.

```
parallel 'htseq-count -a 0 -f bam -m intersection-strict -s yes -t
CDS {} Rattus_norvegicus.protein_coding.gtf \
 > {.}.htseq.count' ::: *.Aligned.sortedByCoord.out.bam
```

Note that we tell HTSeq to only count reads mapping to coding parts of genes (-t CDS) rather than to untranslated regions; and we only assign a read to a given gene if it that read is located entirely within the coding sequence of that gene (-m intersection-strict) and does not span the boundary of the CDS.

The output files from the above command will be given the extension '.htseq.count'. The first column in each of these files gives the Ensembl identifiers for the genes, and the second column gives the number of read counts assigned to each gene. We now need to amalgamate the individual files for each library into a single combined read counts file that we will feed into Xtail for the differential translation analysis.

(In this example, my libraries are named Index1, Index2, Index3, and Index4. The below command extracts the first column [gene identifiers] from the first file; then prints the counts columns for each file separated by tabs. We also remove the last 5 lines from the merged file using the head command; as these lines are summary statistics from HTSeq.)

```
# print a header line to our file indicating which libraries are in
which columns
printf '%s\t%s\t%s\t%s\t%s\n' "GeneID" "Index1" "Index2" "Index3"
"Index4" > mRNA_readcounts_amalgamated.txt

# print the selected columns
paste *.Aligned.sortedByCoord.out.htseq.count \
| awk '{ print $1 "\t" $2 "\t" $4 "\t" $6 "\t" $8 }' | \
head -n -5  >> mRNA_readcounts_amalgamated.txt
```

If you are dealing with more than 4 libraries, you simply append the appropriate column numbers to the awk command above
(e.g. for 6 files, use: `print $1 "\t" $2 "\t" $4 "\t" $6 "\t" $8 "\t" $10 "\t" $12`).

Differential translation analysis with Xtail

Now we are ready to use Xtail to perform the differential translation analysis. Open up an R session, set your working directory to the directory where you have saved the counts file. Load the Xtail library and read in your amalgamated counts file:

```
setwd("/mnt/sdf/MuLV/differential_expression")
library(xtail)

mRNA_readcounts <- read.table("mRNA_readcounts_amalgamated.txt",
   header=T,row.names=1,stringsAsFactors = FALSE)
```

Next, we assign individual columns in the counts file as being either RNAseq libraries or RiboSeq libraries (RPF = ribosome protected fragments). In this example, my RiboSeq libraries are in the first, third, fifth, and seventh columns; while my RNASeq libraries are in the second, fourth, sixth, and eighth columns.

```
rna <- mRNA_readcounts[,c(2,4,6,8)]
rpf <- mRNA_readcounts[,c(1,3,5,7)]
```

We also need to specify which libraries are mocks and which are virus-infected. In this example, the first pair of samples (columns 1+2) are mocks, while the second pair (columns 3+4) are infected; etc.

```
condition <- c("mock","infected","mock","infected")
```

Now we run xtail using the xtail() function:

```
results <- xtail(rna,rpf,condition)
```

This command will take a while to run. If you would like more information on what the Xtail function is doing, and what options are available, just type ‘`?xtail`’ on the R command line.

Note that, by default, the xtail() function compares the second condition (here, "infected") against the first (here, "mock"). Therefore, in this example, the resulting log fold change (logFC) values will be positive where a gene is more highly translated in infections vs mocks, and negative where a gene is translated at a lower level in infections vs mocks.

When the xtail() function finishes, you can print a short summary of the results:

```
summary(results)
```

This will tell us how many genes were used for the analysis, and how many of them have an adjusted pvalue < 0.1.

We can output the full list of genes, with their log fold changes and p-values to a tab-separated file using the write.xtail() function:

```
write.xtail(results,"xtail.results",quote=F,sep="\t")
```

Additionally, we can make plots showing the overall distribution of logFC values, expression ratios, and pvalues:

```
plotFCs(results,cex=0.6)
plotRs(results,cex=0.6)
volcanoPlot(results)
```

GO term enrichment using topGO

We can now perform GO term enrichment testing on differentially translated genes, to check if there are specific biological functions which are over-represented among those genes.

First, use the xtail results file to select genes which are differentially translated at a given adjusted p-value. For this example, we will choose all genes with adjusted p-values < 0.05. The adjusted p-values are given in the eighth column (labelled “pvalue.adjust") of the xtail results. We will separately extract upregulated (log2FC > 0) and downregulated (log2FC < 0) genes, to test both sets independently for GO term enrichment.

```
awk '{if ($8<0.05&&$6>0) print $1}' \
  xtail.results >  xtail.results.upregulated

awk '{if ($8<0.05&&$6<0) print $1}' \
  xtail.results >  xtail.results.downregulated

#remove the header lines from both files
sed -i -e "1d" xtail.results.upregulated
sed -i -e "1d" xtail.results.downregulated
```

To test for GO term enrichment, we also want to specify a "background" gene set which tells us how frequently each GO term occurs across the genome. Therefore, let's extract the full list of gene accessions from our amalgamated readcounts file:

```
awk '{print $1}' mRNA_readcounts_amalgamated.txt \
 > HTSeq_counted_genes.txt

#remove the header line
sed -i -e "1d" HTSeq_counted_genes.txt
```

Now, start up an R session and load topGO:

```
setwd("/mnt/sdf/MuLV/differential_expression")
library(topGO)
```

Specify our background gene set (which topGO refers to as the 'Gene Universe'):

```
geneID2GO <- readMappings(file = "HTSeq_gene_GO_terms.txt")
geneUniverse <- names(geneID2GO)
```

Read in our list of DE genes (starting with upregulated genes):

```
genesOfInterest <- read.table("xtail.results.upregulated",
header=FALSE,row.names=NULL)

genesOfInterest <- as.character(genesOfInterest[,1])

geneList <- factor(as.integer(geneUniverse %in%
genesOfInterest))

names(geneList) <- geneUniverse
```

Now build the topGOdata object containing the gene lists and GO annotation data:

```
myGOdata <- new("topGOdata",
description="DE_genes_GO_enrichment", ontology="BP",
allGenes=geneList,  annot = annFUN.gene2GO, gene2GO = geneID2GO)
```

There are a number of startistical tests available through topGO for enrichment testing. See see the available tests, use the whichTests() function. Here, we will use Fisher's Exact Test.

```
resultFisher <- runTest(myGOdata, algorithm="weight01",
statistic="fisher")
```

Use the GenTable() function to generate a summary table with the results of the X top GO terms (specified by the topNodes parameter). Here, we will print the top 20 terms:

```
allRes <- GenTable(myGOdata, classic = resultFisher,
  orderBy = "resultFisher", ranksOf = "classicFisher",
  topNodes = 20)
```

Finally, write the results to an output file:

```
write.table(allRes,
file="upregulated_genes_GO_enrichment.txt",quote=F,sep="\t")
#last column is unadjusted pval
```

Now you can repeat the process for downregulated genes.