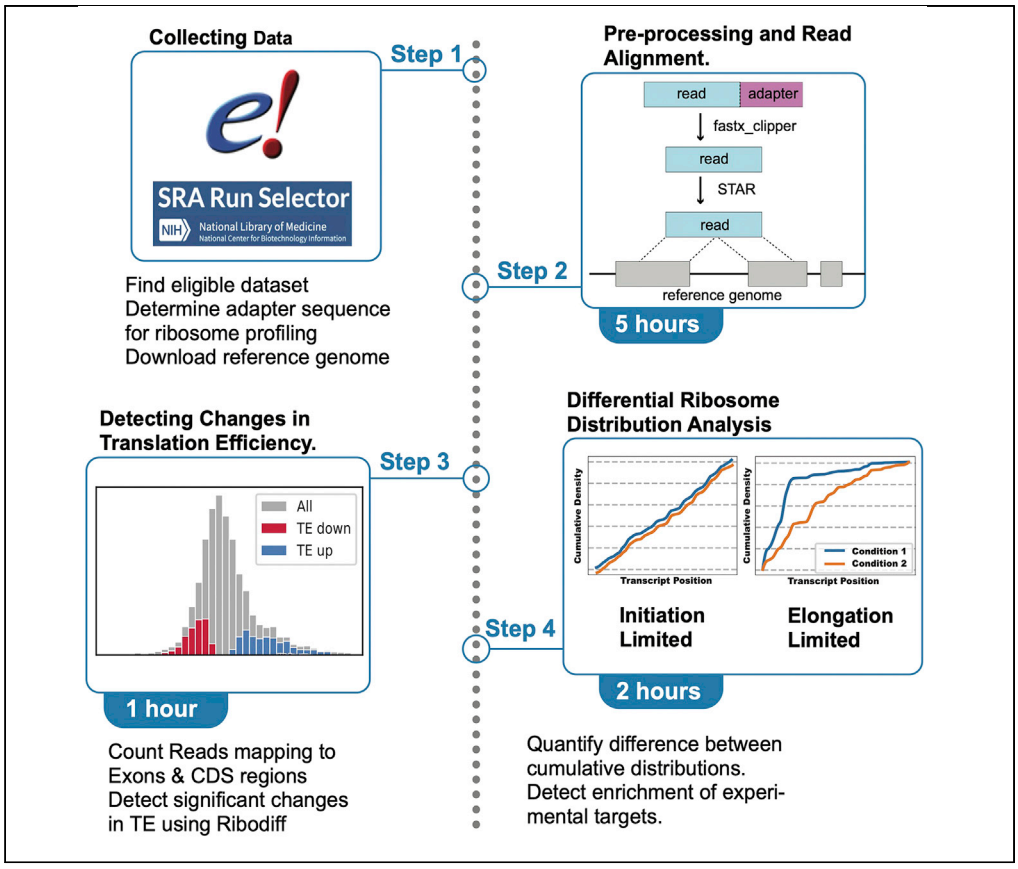
STAR Protocols

CellPress
OPEN ACCESS

Protocol

End-to-end pipeline for differential analysis of pausing in ribosome profiling data



Ribosome profiling is a powerful technique which maps the distribution of ribosomes along mRNAs to analyze translation genome-wide. Ribosome density can be affected by multiple factors, such as changes to translation initiation or elongation rates. We describe the application of a metric for identifying genes rate-limited by these rates by analyzing the relative distribution of ribosome footprints along transcripts. This protocol also details two sample analyses comparing gene translation efficiencies and the distribution of ribosome densities on downloadable datasets.

Publisher's note: Undertaking any experimental protocol requires adherence to local institutional guidelines for laboratory safety and ethics.

Keegan Flanagan, Wanxin Li, Ethan J. Greenblatt, Khanh Dao Duc

ethan.greenblatt@ubc.ca (E.J.G.)
kdd@math.ubc.ca (K.D.D.)

Highlights

Pipeline for measuring translation efficiency using ribosome profiling data

Quantification of differential ribosome distributions along mRNAs

Determine whether changes are consistent with altered initiation or elongation

Flanagan et al., STAR Protocols 3, 101605
September 16, 2022
https://doi.org/10.1016/j.xpro.2022.101605

STAR Protocols

CellPress
OPEN ACCESS

Protocol

End-to-end pipeline for differential analysis of pausing in ribosome profiling data

Keegan Flanagan,[1,2] Wanxin Li,[3,4] Ethan J. Greenblatt,[1,5,*] and Khanh Dao Duc[2,4,6,*]

[1]Department of Biochemistry and Molecular Biology, University of British Columbia, Vancouver, BC, Canada

[2]Department of Mathematics, University of British Columbia, Vancouver, BC, Canada

[3]School of Computer Science, University of Waterloo, Waterloo, ON, Canada

[4]Department of Computer Science, University of British Columbia, Vancouver, BC, Canada

[5]Technical contact

[6]Lead contact

*Correspondence: ethan.greenblatt@ubc.ca (E.J.G.), kdd@math.ubc.ca (K.D.D.)
https://doi.org/10.1016/j.xpro.2022.101605

SUMMARY

Ribosome profiling is a powerful technique which maps the distribution of ribosomes along mRNAs to analyze translation genome-wide. Ribosome density can be affected by multiple factors, such as changes to translation initiation or elongation rates. We describe the application of a metric for identifying genes rate-limited by these rates by analyzing the relative distribution of ribosome footprints along transcripts. This protocol also details two sample analyses comparing gene translation efficiencies and the distribution of ribosome densities on downloadable datasets.

For complete details on the use and execution of this protocol, please refer to Flanagan et al. (2022).

BEFORE YOU BEGIN

Here we detail a protocol to quantify changes in ribosome density distributions along transcripts under two different conditions. The protocol relies on a metric that identifies changes in the rate limiting step of translation. It uses the fact that there is a distinct difference in the distribution of read densities from initiation limited and elongation limited transcripts (Figure 1). Our protocol can also be adapted to identify transcripts which have leaky translation termination or alternative translation initiation sites. When translation is limited by the minimum elongation rate, ribosomes collide and become backed up behind a specific stall site, since the ribosomes are being added to the transcript faster than they can travel along it. This causes the reads detected by ribosome profiling to predominantly come from the 5′ end of the transcript. The reads from initiation limited transcripts are comparatively evenly distributed along the length of the transcript, where sites with low elongation rates only cause brief spikes in read density. This behavior has been thoroughly characterized using stochastic models of translation, such as the inhomogeneous l-TASEP model (Erdmann-Pham et al. 2020, 2021), and has been observed in ribosome profiling experiments done on cells that have undergone elongation limitation inducing treatments (Woolstenhulme et al., 2015; Fradejas-Villar et al., 2017). Our metric is determined using the K-S statistic, which is traditionally calculated as part of the K-S test, a well-documented test for determining if two distributions are statistically equivalent. We calculate the K-S statistic as the maximum distance between two smoothed cumulative distribution created from the ribosome read distributions. Simulations based on the inhomogeneous l-TASEP model have demonstrated that our metric is highly sensitive to changes from elongation limited translation to elongation limited translation, and we have used it to correctly

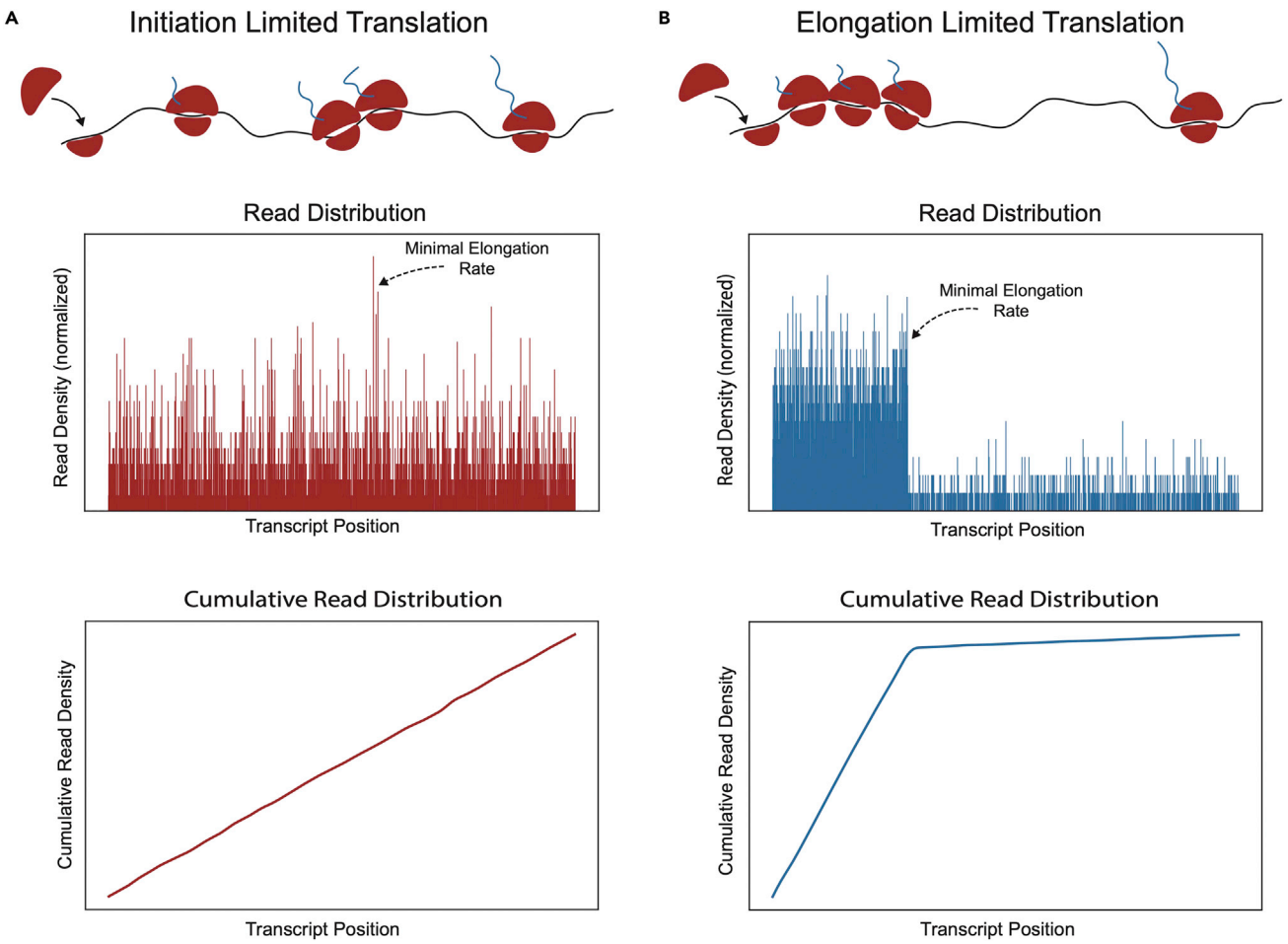


Figure 1. Illustration of translation dynamics and resulting read distributions for initiation rate-limited translation

Read densities from initiation limited transcripts are similar across the entire transcript, whereas under elongation limitation reads preferentially come from the 5′ end prior to a stalling site (Erdmann-Pham et al., 2020; Woolstenhulme et al., 2015). This causes the cumulative distribution of reads to have a characteristic kinked shape. The K-S statistic is calculated by comparing the maximum distance between the two cumulative distributions. As such, the K-S statistic will be relatively high when comparing distributions of transcripts which are initiation limited under one condition and elongation limited in the other. Read density data were created using simulations based on the inhomogeneous l-TASEP model (Erdmann-Pham et al., 2021).

identify genes that are known to experience elongation limited translation under specific treatments (Flanagan et al., 2022). The protocol below details the computation of the metric starting from the processing of raw ribosome profiling data. A differential analysis of translation efficiency between datasets is also detailed, so that changes in translation efficiency and rate limitation can be compared.

Pre-processing and alignment setup

Timing: 10 min

The steps outlined in the Processing raw ribosome profiling and RNA sequence data section require several different command line tools. All of these tools can be easily downloaded using Miniconda, an open-source package manager (or its larger version Anaconda).

1. Install Miniconda using the following commands:

{Bash}

```
# For Linux:
> curl -O https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
> sh Miniconda3-latest-Linux-x86_64.sh
# For Mac:
> curl -O https://repo.anaconda.com/miniconda/Miniconda3-latest-MacOSX-x86_64.sh
> sh Miniconda3-latest-MacOSX-x86_64.sh
```

Note: Additional information on installing Miniconda and instructions for installation for windows operating systems can be found at https://docs.conda.io/en/latest/miniconda.html.

2. Set up the Bioconda channel by running:

{Bash}

```
> conda config –add channels defaults
> conda config –add channels bioconda
> conda config –add channels conda-forge
```

3. Download and install the following packages using conda:

{Bash}

```
> conda install -c bioconda star
> conda install -c bioconda fastx_toolkit
> conda install -c bioconda subread
> conda install -c bioconda sra-tools
> conda install samtools
> conda install bowtie2
```

Installing RiboDiff

Timing: 20 min

RiboDiff is a software package that can determine if there are significant protein translation efficiency changes between two datasets using ribosome profiling and RNA-seq data (Zhong et al., 2017). RiboDiff is built in Python2 and will require a Python 2 conda environment to function.

4. Install the RiboDiff dependencies within a python 2 conda environment:

{Bash}

```
> conda create –name py2 python=2.7

> conda activate py2

> conda install -c bioconda ribodiff
```

⚠ CRITICAL: Installing RiboDiff with bioconda only installs the dependencies of RiboDiff. It does not install the complete program.

5. Clone the RiboDiff GitHub repository:

```
> git clone https://github.com/ratschlab/RiboDiff.git
```

6. Complete the RiboDiff installation by entering the newly created RiboDiff repository and installing the program with Python2:

{Bash}

```
> cd RiboDiff

> python2 setup.py build

> python2 setup.py install –user
```

7. Exit the Python 2 conda environment:

```
> conda deactivate
```

Plastid and python environment preparations

Timing: 20 min

Plastid is a Python library which contains a variety of tools for genomics and sequencing and is available on all operating systems (Windows, Linux, Mac OSX). This library will be used extensively within the Creating count arrays section. The Plastid Python library is available as an Anaconda package library and can be easily installed within a Conda environment using Miniconda. Other necessary Python packages can be subsequently installed within the Plastid environment.

8. Install Miniconda and setup Bioconda on your system if you have not already (see the pre-processing and alignment setup section).
9. Create a conda environment with plastid installed:

```
> conda create -n plastid plastid
```

10. Enter the new conda environment:

```
> conda activate plastid
```

11. Plastid is designed to work with an older version of Biopython and is incompatible with newer versions. Therefore, it is necessary to downgrade Biopython by running:

```
> conda install biopython==1.76
```

12. Install Jupyterlab within the Plastid environment:

```
> conda install -c conda-forge jupyterlab
```

13. Install the multiprocess Python package using conda:

```
> conda install multiprocess
```

R environment preparations

Timing: 30 min

The determining p-site offsets section of this protocol requires the use of an R-package called riboWaltz. This section must be installed within R directly from GitHub using the devtools suite of R-packages.

14. Install R and R studio using the instructions from this link https://www.r-project.org/ and this link https://www.rstudio.com/products/rstudio/download/ respectively.
15. Open R-studio and install the devtools suite of packages using R's install.packages function and then load devtools into the R session using the library command:

```
{R}
> install.packages("usethis")
> install.packages("devtools")
> library(devtools)
```

16. Install riboWaltz directly from GitHub using devtools' install_github function. Troubleshooting 1.

```
{R}
> install_github("LabTranslationalArchitectomics/riboWaltz",
  dependencies = TRUE, build_opts = c("-no-resave-data", "-
  no-manual"))
```

RNA sequencing and ribosome profiling data

This protocol is designed to use fastq files collected from mRNA sequencing and ribosome profiling experiments as input. The protocol is split into three major sections, a data pre-processing and genome alignment section and two analysis sections. The pre-processing and alignment section

will require a genome annotation GTF file, a complete genome assembly FASTA file, and a non-coding RNA assembly FASTA file. We recommend Ensembl, RefSeq, GenBank, and the UCSC Genome Browser as viable sources for these reference files. It is also necessary to know the adapter sequence used during the preparation of the ribosome profiling libraries, so that it can be removed from the ribosome profiling reads. The analyses sections of this protocol are both used to detect changes between two datasets which are under different experimental conditions. For both datasets, an mRNA sequencing fastq file and a ribosome profiling fastq file will be needed. The detecting changes in translation efficiency section will require at least two replicates. The differential ribosome distribution analysis section will require a comma or tab delimited text file that lists all the genes that are expected to be affected by the difference in condition between the two datasets.

Note: All of the reference genomes and genome annotation files necessary to run the examples found in this protocol can be found through the key resources table and on our OSF project page (OSF: 5qcwk). The example code shown in this protocol was generated using a testing dataset made from a smaller subset of the data collected by Fradejas-Villar et al. (2017). This testing dataset is useful for quickly determining if the pipeline has been properly set up and can be found on our OSF project page. The complete dataset from Fradejas-Villar et al. (2017) can be accessed from SRA: SRP078005.

KEY RESOURCES TABLE

REAGENT or RESOURCE	SOURCE	IDENTIFIER
Deposited data		
Mouse reference genome assembly	(Kent et al., 2002)	UCSC Genome Browser ftp: ftp://hgdownload.cse.ucsc.edu/goldenPath/mm10/bigZips/mm10.fa.gz
Mouse reference genome annotation file	(Kent et al., 2002)	UCSC Genome Browser ftp: ftp://hgdownload.cse.ucsc.edu/goldenPath/mm10/bigZips/genes/mm10.refGene.gtf.gz
Mouse non-coding RNA assembly, release 105	(Howe et al., 2021)	Ensembl ftp: ftp://ftp.ensembl.org/pub/release-105/fasta/mus_musculus/ncrna/Mus_musculus.GRCm39.ncrna.fa.gz
Mouse mRNA sequencing data	(Fradejas-Villar et al., 2017)	SRA Run Selector: https://www.ncbi.nlm.nih.gov/Traces/study/?acc=GSE84112&o=acc_s%3Aa
Mouse ribosome profiling data	(Fradejas-Villar et al., 2017)	SRA Run Selector: https://www.ncbi.nlm.nih.gov/Traces/study/?acc=GSE84112&o=acc_s%3Aa
List of selenocysteine containing mouse genes	(Santesmasses et al., 2020)	NA
Software and algorithms		
STAR(2.7.9a)	(Dobin et al., 2013)	https://github.com/alexdobin/STAR
RSEM(1.3.3)	(Li and Dewey, 2011)	https://deweylab.github.io/RSEM/
Subread/featureCounts (2.0.1)	(Liao et al., 2014)	http://subread.sourceforge.net/
FASTX-Toolkit (0.013)	(Gordon, 2010)	http://hannonlab.cshl.edu/fastx_toolkit/
ncbi/sra-tools (2.8.0)	NA	https://trace.ncbi.nlm.nih.gov/Traces/sra/sra.cgi?view=software
bowtie2 (2.4.4)	(Langmead and Salzberg, 2012)	http://bowtie-bio.sourceforge.net/bowtie2/index.shtml
plastid (0.4.8)	(Dunn and Weissman, 2016)	https://plastid.readthedocs.io/en/latest/index.html
R (>=3.5.0)	NA	https://www.r-project.org/
Jupyterlab (3.2.4)	(Kluyver et al., 2016)	https://github.com/jupyterlab/jupyterlab
Jupyter	(Kluyver et al., 2016)	https://github.com/jupyter
riboWaltz (1.2.0)	(Lauria et al., 2018)	https://github.com/LabTranslationalArchitectomics/riboWaltz
NumPy (1.21.4)	(Harris et al., 2020)	https://github.com/numpy/numpy
Pandas (1.3.4)	(Reback et al., 2022)	https://github.com/pandas-dev/pandas
Matplotlib (3.5.0)	(Hunter, 2007)	https://matplotlib.org/stable/index.html

(*Continued on next page*)

Continued

REAGENT or RESOURCE	SOURCE	IDENTIFIER
Devtools (2.4.3)	(Wickham et al., 2022)	https://github.com/r-lib/devtools
RiboDiff (0.2.1)	(Zhong et al., 2017)	https://github.com/ratschlab/RiboDiff
Python (>=3.8.1)	(Van Rossum and Drake, 2009)	https://www.python.org/
Other		
Hardware: AMD Ryzen Threadripper 2950X 16-Core Processor, 128 GB RAM, and Ubuntu version 20.04.1	NA	NA

MATERIALS AND EQUIPMENT

- Data - See the RNA sequencing and ribosome profiling data section from before you begin.
- Hardware - This protocol can become very computationally expensive during the alignment steps within the processing raw ribosome profiling data and RNA-sequence data section. How computationally expensive will depend on the size of the genome that the reads are being aligned to. It is recommended that the raw data processing sections be run on a high-performance workstation or supercomputer cluster.
- Processing raw ribosome profiling data and RNA-sequence data section. High performance workstation or cluster - Memory: 64 GB required, 128 GB recommended; Processors: 8 required, 16 recommended.
- All other sections: Memory: 16 GB required, 32 GB recommended; Processors: 1 required, 8+ recommended.
- Software - See the pre-processing and alignment setup, Plastid and python environment and R environment preparation sections of before you begin and the key resources table for software requirements. This protocol was originally tested on a high-performance workstation running the Ubuntu 20.04 Linux operating system.

STEP-BY-STEP METHOD DETAILS

Processing raw ribosome profiling data and RNA-sequence data

Timing: 5 h

The analyses detailed in this protocol are meant to be used to make comparisons between two datasets. These two datasets will be referred to as "condition1" and "condition2" in the protocol's example code. This section is an adaptation of a protocol by Ingolia et al. (2012) and includes details on the pre-processing of the ribosome profiling data, the generation of genome indices, and the alignment to a reference genome. Troubleshooting 2. The ribosome profiling data are aligned to both the genome and the transcriptome to facilitate downstream processing steps.

Note: Ribosome profiling data will often be referenced by the acronym "RPF" while mRNA sequencing data will be referenced by the shortened "RNA".

Note: The example code found below assumes the user is using the example references and data within the "Datasets/reference_files" and "Datasets/testing_fastq_files" folders on our OSF repository. This code should be run using the Datasets folder as a working directory.

⚠ CRITICAL: The testing fastq files found on our OSF page have already had their adapter sequences trimmed. Therefore, step 1 of this protocol must be skipped when working with these testing files. Failure to skip this step will empty the data in the fastq files and they will need to be re-downloaded from OSF.

1. Trim adapter sequences off of ribosome footprinting reads using fastx_clipper. This command will need to be run for each ribosome profiling sample you are analyzing:

{Bash}

```
> fastx_clipper -Q33 -a TGGAATTCTCGGGTGCCAAGG -l 25 -c -n -i <untrimmed_RPF.fastq>
-o condition1_RPF_1.trimmed.fastq
```

   a. **-Q:** Minimum Phred quality score to keep, set to 33 in the example.
   b. **-a:** The adapter sequence used during the ribosome profiling experiment to be clipped. Set as "TGGAATTCTCGGGTGCCAAGG" in example. This input must be changed to the adapter sequence used during the ribosome profiling experiment.
   c. **-l:** Minimum read length to keep. Set as 25 in example.
   d. **-c:** Discard non-clipped sequences.
   e. **-n:** Keep nucleotides with unknown sequences.
   f. **-i:** Name of input file to be clipped. Set as "<untrimmed_RPF.fastq>" in example.
   g. **-o:** Preferred name of output file. Set as "condition1_RPF_1.trimmed.fastq" in example.

2. Generate a bowtie2 index for the non-coding RNA within a new directory:

{Bash}

```
> mkdir ncrna_indices

> cp reference_files/assembly.ncrna.fa ncrna_indices

> cd ncrna_indices

> bowtie2-build assembly.ncrna.fa ncrna_index

> cd ..
```

   a. **-f:** Specify that the command input files will be fasta/fastq files.
   b. **<example.ncrna.fa>:** A positional argument specifying a fasta file or list of fasta files containing the references sequences to be aligned to. Set as "example.ncrna.fa" in example.
   c. **<ncrna_index>:** A positional argument specifying a prefix that will be appended to the start of all output files. Set as "ncrna_index" in example.

3. Remove reads which mapped to ncRNA from the fastq files using bowtie2 and then save the resulting files in a new directory. This step greatly increases the speed of the final alignment step by removing any ribosome profiling reads that align to non-coding RNA. Again, this command will need to be run for each ribosome profiling sample you are analyzing:

{Bash}

```
> mkdir norrna_fastq_files

> bowtie2 -L 23 -un=norrna_fastq_files/condition1_RPF_1.norrna.fastq -x ncrna_indices/
ncrna_index testing_fastq_files/condition1_RPF_1.trimmed.fastq > outputtemp.sam
```

   a. **-L:** Set seed substring length. Set to "23" in example.
   b. **–un:** The path which unpaired reads will be written to. Set to "condition1_RPF.norrna.fastq" in example. For the purpose of this protocol, this is the main output we are looking for in this step.
   c. **-x:** Path to the reference genome index generated in step 2. Set to "ncrna_indices/ncrna_index" in example.
   d. **<condition1_RPF.trimmed.fastq >** Positional argument which specifies the file to be aligned to the ncRNA. Set to "testing_fastq_files/condition1_RPF_1.trimmed.fastq" in example. This will be the same file which was trimmed in step 1.
   e. **<outputtemp.sam>:** The name of the output file. Set to "outputtemp.sam" in example. For the purpose of this protocol, this output is not important and is written to a temporary output file.

4. Generate a STAR genome index within a new directory. Troubleshooting 3.

{Bash}

```
> mkdir star_indices

> STAR –runThreadN 8 –runMode genomeGenerate –genomeDir star_indices –genomeFastaFiles reference_files/assembly.fa –sjdbGTFfile reference_files/annotation.gtf –genomeSAindexNbases 13
```

a. **–runThreadN:** The number of threads to be used to generate the indices. Set to 8 in example. Should be set at or below the number of available cores on the system.
b. **–runMode:** The type of process STAR is running. Set to "genomeGenerate" in example.
c. **–genomeDir:** The path to a directory where the genome indices will be stored. Set as "star_indices" in example.
d. **–genomeFastaFiles:** The path to a fasta file containing the complete genome assembly for the organism of study. Set as "assembly.fa" in example.
e. **–sjdbGTFfile:** The path to a gtf file containing a genome annotation file for the organism of study. Set to "annotation.gtf" in example.
f. **–genomeSAindexNbases**: The length of the SA pre-indexing string in base pairs. Should be chosen depending on genome size using the equation min(14, log2(length of genome)/2 -1). Set as "13" in example.

5. Align the ribosome profiling data to the reference genome. This command will need to be repeated for each sample you are analyzing:

Note: Each time this step is run it will output two files. One that is aligned to the genome and has the format "condition_RPF_#_Aligned.sortedByCoord.out.bam" and one that is aligned to the transcriptome and has the format "condition_RPF_#_Aligned.toTranscriptome.out.bam."

{Bash}

```
> mkdir star_alignments

> STAR –genomeDir star_indices –runThreadN 8 –readFilesIn norrna_fastq_files/condition1_RPF_1.norrna.fastq –outFileNamePrefix star_alignments/condition1_RPF_1_ –outSAMtype BAM SortedByCoordinate –quantMode TranscriptomeSAM
```

a. **–genomeDir:** A path to a directory where STAR genome indices are stored. Set to "star_indices" in example.
b. **– runThreadN:** The number of threads to be used for the gene mapping. Set to 8 in example. Should be set at or below the number of available cores on the system.
c. **–readFilesIn:** The path and name of the file containing the sequences to be mapped to the reference genome. Set to "condition1_RPF.norrna.fastq" in example.
d. **–outFileNamePrefix:** A prefix that will be appended to the start of all output file names. Set to "condition1_RPF_" in example.
e. **–outSAMtype:** The type of SAM/BAM file which will be outputted. Set to "BAM SortedByCoordinate" in example.
f. **–quantMode:** The types of different quantifications requested. In the example it is set to "TranscriptomeSAM" so that SAM/BAM alignments to the transcriptome are outputted into separate files.

6. Align the mRNA sequencing data to the reference genome. This command will need to be repeated for each sample you are analyzing:

{Bash}

```
> STAR –genomeDir star_indices –runThreadN 8 –readFilesIn testing_fastq_files/condition1_RNA_1.fastq –outFileNamePrefix star_alignments/condition1_RNA_1_ –outSAMtype BAM SortedByCoordinate
```

See previous step for list of arguments.

7. Separate the genome alignment and transcriptome alignment files into two different folders:

{Bash}

```
> cp star_alignments/*Aligned.sortedByCoord.out.bam testing_genome_alignments

> cp star_alignments/*Aligned.toTranscriptome.out.bam testing_transcriptome_alignments
```

8. Create BAM index files for all of the genome alignment files generated in the last 2 steps:

{Bash}

```
> cd testing_genome_alignments

> samtools index condition1_RPF_1_Aligned.sortedByCoord.out.bam

> samtools index condition1_RNA_1_Aligned.sortedByCoord.out.bam

> cd ..
```

Pause point: Their will be many different intermediate outputs from these first few steps. The only files that are needed for the rest of the protocol are the BAM files outputted by steps 5 and 6 and the associated BAM index files generated in step 8. For each ribosome profiling sample there will be two BAM files, one for the genome alignment and one for the transcriptome alignment. The RNA samples will have only one BAM file each which will be for the genome alignment.

Detecting changes in translation efficiency

Timing: 1 h

Translation efficiency is defined as the ratio of normalized ribosome footprint reads to RNA-seq reads for individual transcripts. We use featureCounts to count the number of ribosome profiling reads and mRNA sequencing reads that map to CDS regions and exons respectively for both of our datasets. Then, a short Python script is used to convert the output of featureCounts into suitable inputs for RiboDiff. Finally, RiboDiff is used to quantify the changes in translation efficiency for each gene and determine the significance of the change.

⚠ CRITICAL: In order to run RiboDiff it is necessary for there to be at least 2 replicates.

Note: This section can be tested using the example GTF file found in the "Datasets/reference_files" folder and the example Bam files found in the "Datasets/testing_genome_alignments" folder provided on our OSF repository. The following commands should be run using the Datasets folder as a working directory.

9. Use featureCounts to count the individual number of reads that align to specific genomic features. We are interested in reads that align to exons for RNA-seq experiments and reads that align to CDS (protein coding) regions for ribosome profiling experiments:

{Bash}

```
> mkdir TE_results

> featureCounts -T 8 -t CDS -a reference_files/annotation.gtf -o TE_results/counts_condition1_RPF_1.txt
testing_genome_alignments/condition1_RPF_1_Aligned.sortedByCoord.out.bam
```

a. **-T:** The number of threads to be used when counting features. Set to 8 in example. Should be set at or below the number of available cores on the system.
b. **-t:** The specific feature type to be used for read counting. Set to "CDS" in example.
c. **-a:** The path and name of a GTF genome annotation file for the organism of choice. Set to "`reference_files/annotation.gtf''` in example.
d. **-o:** The preferred name of the output file. Set to "counts_condition1_RPF.txt" in example.
e. **<testing_genome_alignments /condition1_RPF_1_Aligned.sortedByCoord.out.bam >** Positional argument for the alignment files to be inputted into featureCounts. The reads in the file must be aligned using the same genome annotation file as the one specified in "-a".

⚠ CRITICAL: When running this step on the mRNA sequencing data, "-t CDS" should be changed to "-t exon". This step must be run for all the genome alignment files from both datasets before proceeding to step 10.

10. Use the process_counts.py file located in the "Python_scripts" folder on our OSF project page to automatically take all of the count files generated by featureCounts and arrange them into the correct format for RiboDiff. This Python script should be run within the directory containing the output from featureCounts.

{Bash}

```
> cp <path/to/process_counts.py> TE_results

> cd TE_results

> python3 process_counts.py -path-to-counts . -treated-prefix condition1 -control-prefix
condition2
```

a. **–path-to-counts:** The path to a folder which contains the count arrays outputted by feature counts.
b. **–treated-prefix:** A prefix that is contained within the name of the count files for the condition1 data.
c. **–control-prefix:** A prefix that is contained within the name of the count files for the condition2 data.

11. Currently RiboDiff has not been updated to work with Python3+. As such, we need to activate the py2 environment created in installing RiboDiff section in order to run the program:

```
> conda activate py2
```

12. Run the RiboDiff program using python 2:

{Bash}

```
> python2 <path/to/RiboDiff/scripts/TE.py> -e experimental_design.csv -c raw_read_count.txt -o ribo_output.txt -p 1
```

a. **-e:** A text file describing the format of the experiment. This should be set to the output of the process_counts.py script. Set to "experimental_design.csv" in example.
b. **-c:** A text file containing the count data. This should be set to the output of the process_counts.py script. Set to "raw_read_count.txt" in example.
c. **-o:** Preferred name for the tab delimited text file which will be outputted by RiboDiff. Set to "ribo_output.txt" in example.
d. **-p:** Make plots to show the data and results (Figure 2).

13. Deactivate the py2 environment and change working directory back to Datasets:

{Bash}

```
> conda deactivate

> cd ..
```

Determining P-site offsets

Timing: 1 h

Ribosome protected fragments are often a variety of different lengths, so we do not know the exact location of the ribosome along the transcript from the ribosome profiling data. We will map each footprint to a precise location along a transcript using P-site offsets. The P-site offset is the distance between the P-site of the ribosome and the extremities (the 5' or 3' end) of the read. By estimating the most likely P-site offset for each read length we can accurately determine the position of the ribosome along the transcript. For this analysis, P-site offsets were determined using the riboWaltz R package Lauria et al. (2018).

Note: This section can be run using the GTF file found in the "Datasets/reference_files" folder and the Bam files found in the "Datasets/testing _transcriptome_alignments" folder on our OSF repository. An example R script containing the code for this section can be found in the "R_scripts" folder on OSF and our GitHub repository. This example R script should be run while using the "R_scripts" folder as a working directory.

14. Load the riboWaltz package into your R session.

```
> library(ribowaltz)
```

15. Load the GTF annotation file used during genome alignment into R using riboWaltz's create_annotation function. Troubleshooting 4.

{R}

```
> df = create_annotation(gtfpath = "<path/to/annotation.gtf>")
```

16. Load the bam files from the transcriptome alignment into R using the bamtolist function. Note that this function operates by automatically loading up all of the bam files within a folder specified by the user:

{R}

```
> bam_list = bamtolist("<path/to/transcriptome_alignments_folder>", annotation = df)
```

⚠ CRITICAL: This step can be RAM intensive and may cause R to crash if insufficient RAM is available. If R consistently crashes at this step, check to ensure that the machine being used meets the 16 GB RAM requirement stated in the Materials and equipment section and close additional processes running on the machine.

17. Calculate the P-site offsets for all of the bam files by running the psite() function:

```
> offsets = psite(data = bam_list, start = FALSE)
```

Note: In this example the "start" argument is set to "FALSE". Setting this argument to false directs riboWaltz to use the second to last codon on a transcript as a reference codon instead of the first codon. This was set to false because the datasets originally used to test this protocol did not exhibit initiation peaks, so it was unsuitable to use the first codon as a reference for p-site offsetting. If this protocol is used on data collected from cell cultures which demonstrate an initiation peak, it may be beneficial to alter this argument to "TRUE".

18. Extract the important information from the output of psite by creating a new dataframe using R's built-in subset function. The values needed from the output are length, which should be saved in a column called length, and corrected_offset_from_3, which should be saved in a column called p_offset. Save the newly created dataframe as a tab delimited text file using write.table:

{R}

```
> samples = unique(offsets$sample)
> for (i in 1:length(samples)) {
  > sam_offs = subset(offsets, sample == samples[i],
  select = c("length", "corrected_offset_from_3"))
  > colnames(sam_offs) = c("length", "p_offset")
  > write.table(sam_offs, paste("<path/to/save/p_site_offsets/>",
    samples[i],"_p-site-offsets", sep=""),sep = "\t",
    row.names = FALSE)
}
```

Creating count arrays

Timing: 1 h

Before the analysis of translation limitation can begin the data from the ribosome profiling experiments must be organized into count arrays. Count arrays are basically lists that record the number of reads which map to each base pair or codon position along a transcript. The count arrays will be created inside of a Jupyter notebook which is running inside of the Plastid Conda environment set up in the Plastid and Python environment preparations section. Using Plastid to create the count arrays will allow for important adjustments to be made to the data such as applying the p-site offsets made in the determining p-site offsets section and sub-setting the data to only look at the coding

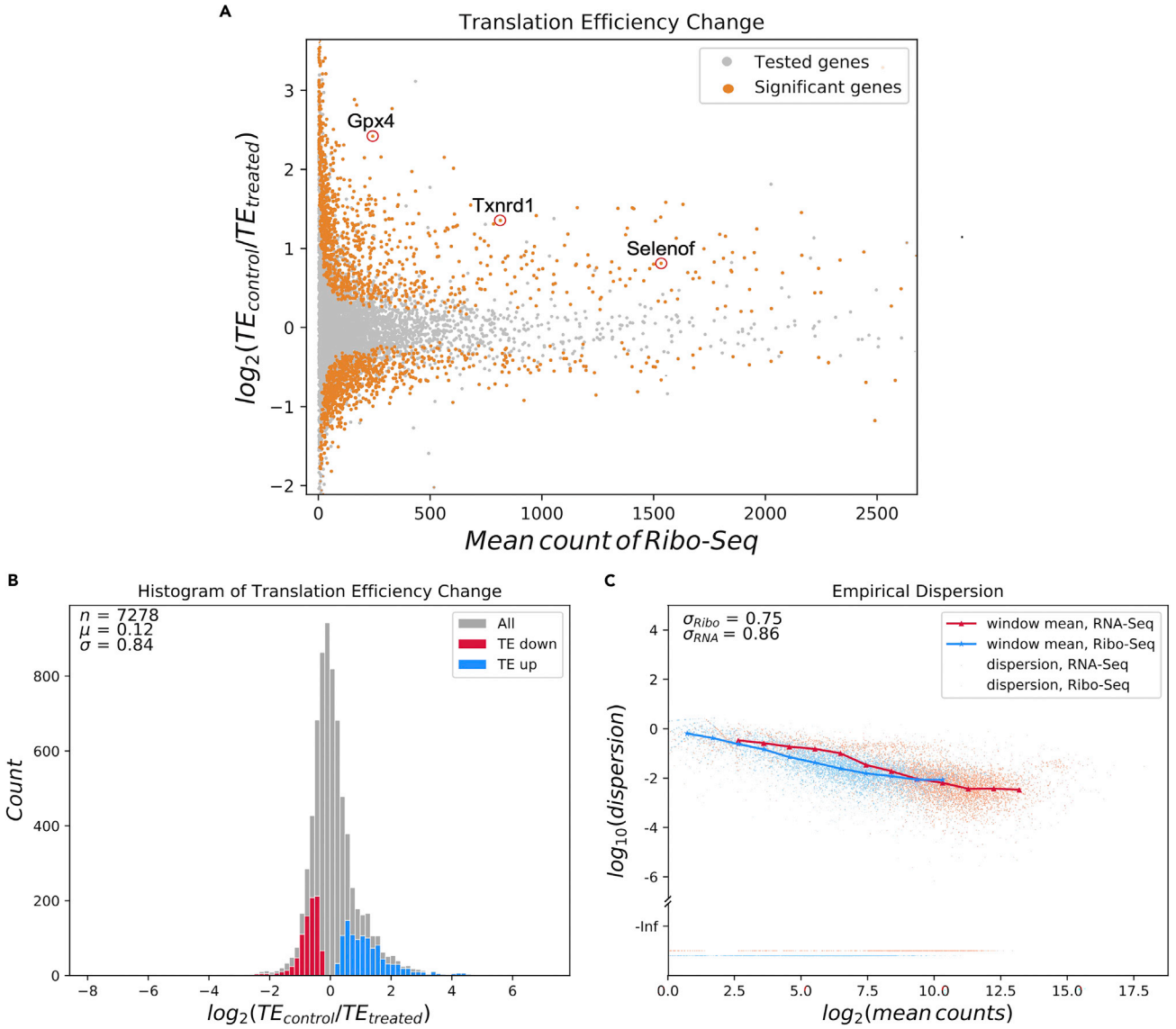


Figure 2. RiboDiff outputs

(A) Scatterplot comparing the log transformed translation efficiency ratios vs the mean read counts from the ribosome profiling data. Genes found to have significant changes in translation efficiency are colored yellow while non-significant genes are colored in gray. Genes exhibiting high K-S statistics are named and circled in red.

(B) Histogram showing the distribution of log transformed translation efficiency ratios for all genes. The distribution for genes found to have significantly increased or decreased translation efficiencies are shown in red and blue respectively.

(C) Comparison of dispersion measurements from sequencing data. Scatterplot showing similar amounts of dispersion in RNA-seq and ribosome profiling data, indicating that the use of a single dispersion measurement is sufficient.

regions of the transcripts. The count arrays will be saved as simple csv tables which can be easily incorporated into further analyses in later sections.

Note: This section can be run using the GTF file found in the "Datasets/reference_files" folder, the text files found in the "Datasets/testing_Psite_offsets" folder, and the Bam files found in the "Datasets/testing_genome_alignments" folder on our OSF repository. An example notebook containing the code for this section can be found in the "Notebooks" folder on OSF and

our GitHub repository. This example notebook should be run within the "Notebooks" directory.

⚠ CRITICAL: This section requires that the utilities.py file located on our OSF project page is within the module search path for your python session. This can be done either by saving the utilities.py file within the same directory as your Jupyter notebooks or by appending the path to the folder containing the utilities.py file to your python session path using the following code:

```
> import sys

> sys.path.append("<path/to/Python_scripts>")
```

19. Load in the python libraries and functions necessary for this pipeline. This includes several functions from plastid and the contents of our utilities.py file:

{Python3}

```
> from plastid import BAMGenomeArray, GTF2_TranscriptAssembler, Transcript

> import numpy as np

> import pandas as pd

> from plastid.plotting.plots import *

> import utilities as utils

> import matplotlib.pyplot as plt

> from matplotlib.pyplot import figure

> %matplotlib inline
```

20. Load in the table of P-site offsets created in the determining p-site offsets section using the Pandas function read_csv:

{Python3}

```
> # Load in the P-site offsets for condition 1

> p_offsets_cond1 = pd.read_csv("<path/to/condition1_RPF_1_Aligned.toTranscriptome.out_
p-site-offsets>", sep="\t")

> # Load in the P-site offsets for condition 2

> p_offsets_cond2 = pd.read_csv("path/to/condition2_RPF_1_Aligned.toTranscriptome.out_
p-site-offsets", sep="\t")
```

21. Load in a GTF genome annotation file into python using Plastid's GTF2_TranscriptAssembler function. This function will load in the transcripts as an iterator of Plastid's transcript type objects which we will then convert to a list using Python's list function:

{Python3}

```
> # Load in the transcript information
> transcripts = list(GTF2_TranscriptAssembler(open("<path/to/annotation.gtf>"), return_type=Transcript))
```

22. Load in the Bam file containing the Ribosome Profiling data as a Bam Genome Array using Plastid's BamGenomeArray() function and map the reads to their corresponding P-sites via the VariableThreePrimeMapFactory custom function in utilities.py and Plastid's set_mapping function:

{Python3}

```
> # Load in the alignments from both the condition 1 and condition 2 datasets
> alignments_cond1 = BAMGenomeArray("path/to/condition1_RPF_1_Aligned.sortedByCoord.out.bam")
> alignments_cond2 = BAMGenomeArray("path/to/condition2_RPF_1_Aligned.sortedByCoord.out.bam")
> # Set the P-site offset mappings for both datasets
> alignments_cond1.set_mapping(utils.VariableThreePrimeMapFactory(p_offsets = p_offsets_cond1))
> alignments_cond2.set_mapping(utils.VariableThreePrimeMapFactory(p_offsets = p_offsets_cond2))
```

23. For each transcript object in our list use Plastid's get_counts function to create a numpy array that contains the number of counts at each position in the transcript:

{Python3}

```
> # Initialize two lists to contain the count arrays
> count_arrays_cond1 = []
> count_arrays_cond2 = []
> # Iterate through each transcript in the GTF file and extract the count arrays for that transcript from the alignments.
> for transcript in transcripts:
  > count_arrays_cond1.append(
    transcript.get_counts(alignments_cond1))
  > count_arrays_cond2.append(
    transcript.get_counts(alignments_cond2))
```

24. Once the count arrays have been created the information on CDS regions contained in the transcript type objects can be used to alter the count arrays to only cover the CDS regions:

{Python3}

```
> # Initialize two lists which will contain the start and end positions of the CDS region of each transcript.
> cds_starts = []
> cds_ends = []
> # Iterate through each transcript and add the cds information to the lists
> for transcript in transcripts:
  > cds_starts.append(transcript.cds_start)
  > cds_ends.append(transcript.cds_end)
> # subset each count array to only look at the cds region.
> for i in range(len(count_arrays_cond1)):
  > count_arrays_cond1[i] =
    list(count_arrays_cond1[i][cds_starts[i]:cds_ends[i]])
  > count_arrays_cond2[i] =
    list(count_arrays_cond2[i][cds_starts[i]:cds_ends[i]])
```

25. Use the add_gene_ids function from utilities.py to append the transcript ID and gene ID of each transcript to the start of the count array:

{Python3}

```
> utils.add_gene_ids(transcripts, count_arrays_cond1)
> utils.add_gene_ids(transcripts, count_arrays_cond2)
```

26. Filter out any count arrays that are of insufficient length or have insufficient read density. In this example, count arrays which were under 200 base pairs in length or which had a read density cut-off below 0.15 reads per base pair were filtered out:

{Python3}

```
> # Initialize two lists to hold the filtered arrays
> filtered_array_cond1 = []
> filtered_array_cond2 = []
> # Iterate through each of the count arrays and save any count arrays that pass our filtering parameters
> for array_1, array_2 in zip(count_arrays_cond1,
  count_arrays_cond2):
    > if len(array_1) > 200 and
      sum(array_1 [2:])/len(array_1 [2:]) > 0.15 and
      sum(array_2 [2:])/len(array_2 [2:]) > 0.15:
        > filtered_array_cond1.append(array_1)
        > filtered_array_cond2.append(array_2)
```

Note: The 0.15 reads per base pair cutoff was determined empirically in Flanagan et al. (2022) using simulations of ribosome profiles with different read densities. Depending on the amount of noise present within the data it may be appropriate to use a larger or smaller cutoff. If K-S statistic values are inconsistent across replicates, a more stringent cutoff should be applied.

27. Save the count arrays to be used in future notebooks. Use the custom save_count_positions function from utilities.py so that the count arrays are saved with a header that describes each column which it is easier to read. Troubleshooting 5.

{Python3}

```
> utils.save_count_positions(filtered_array_cond1,
  "<path/to/save/condition1_1_counts.csv>")
> utils.save_count_positions(filtered_array_cond2,
  "<path/to/save/condition2_1_counts.csv>")
```

Differential ribosome distribution analysis

Timing: 30 min

This section will cover how to determine if the difference between our two datasets induces rate limiting pauses during translation elongation. First, the count arrays from the last section will be loaded into a Jupyter notebook and filtered so that only transcripts with sufficient length and read coverage are analyzed. Then, LOESS smoothing will be performed on the filtered count arrays and the cumulative distributions of the smoothed count arrays will be calculated. These cumulative distributions will be used to calculate the K-S statistic for each gene. All of the genes will then be sorted into 3 separate bins based on whether they have low, medium, or high K-S statistics. The fold enrichment of genes within each bin is then calculated. Finally, Fisher's exact test will be used to determine if the observed enrichment of the target genes in various K-S bins is statistically significant.

Note: This section can be run using the csv files from the "Datasets/testing_count_arrays" folder and the csv file from the "Datasets/condition_targets" folder on our OSF repository. An example notebook containing the code for this section can be found in the "Notebooks" folder on OSF and our GitHub repository. This example notebook should be run within the "Notebooks" directory.

⚠ CRITICAL: This section requires that the utilities.py file located on our OSF project page is within the module search path for your python session. This can be done either by saving the utilities.py file within the same directory as your Jupyter notebooks or by appending the path to the folder containing the utilities.py file to your python session path using the following code:

```
> import sys
> sys.path.append(''<path/to/utilities_folder>'')
```

28. Load in all of the necessary Python packages:

{Python3}

```
> import numpy as np
> import pandas as pd
> import utilities as utils
> import scipy.stats as stats
> from statsmodels.stats.multitest import multipletests
> from decimal import Decimal
> import matplotlib.pyplot as plt
> from matplotlib.pyplot import figure
> %matplotlib inline
> from multiprocess import Pool
```

29. Load in the count arrays for both datasets:

{Python3}

```
> # Load in the count arrays from the condition 1 dataset
> condition1, names_1 = utils.load_count_positions("<path/to/condition1_1_counts.csv>")
> # Load in the count arrays from the condition 1 dataset
> condition2, names_2 = utils.load_count_positions("<path/to/condition2_1_counts.csv>")
```

30. Smooth out the count arrays using LOESS smoothing and calculate the cumulative read distributions. The get_smoothed_array function from the diff_utils.py file is set up to perform Loess smoothing with a window size equal to 5% of the transcript length and calculates the cumulative read distribution. This step can take some time, so it is recommended to use multiprocess' Pool() function to complete this using multiple cores:

{Python3}

```
> # Define the number of processers to use.
> max_pool = 8
> # Iterate through each dataset and calculate the smoothed density array
> with Pool(max_pool) as p:
  > pool_1, pool_2 = list(
      p.imap(utils.get_smoothed_array,
          condition1)
    ), list(
      p.imap(utils.get_smoothed_array,
          condition2)
    )
```

31. For each gene in both datasets, calculate the K-S-statistic as the maximum distance between their smoothed cumulative distributions and then save this K-S-statistic as a list:

{Python3}

```
> # initialize 2 lists to hold the length and K-S statistic for each gene
> ks_list = []
> len_list = []
> # iterate through each transcript in both datasets and calculate the length and K-S Statistic
> for tr_1, tr_2, index in zip(condition1, condition2,
  list(range(len(condition1)))):
    > smoothed_array_1, cumul_1 = pool_1[index]
    > smoothed_array_2, cumul_2 = pool_2[index]
    > ks = max(abs(cumul_1 - cumul_2))
    > ks_list.append(ks)
    > len_list.append(len(tr_1))
```

32. Combine the lists for the gene IDs, gene lengths, and K-S statistics into a pandas dataframe using the Pandas' DataFrame function and rename the columns of the dataframe to something more suitable:

{Python3}

```
> # Create a pandas dataframe of K-S statistics
> ks_table = pd.DataFrame(list(zip(names_1, ks_list,
len_list)))
> # Rename the dataframe columns
> ks_table.columns = ["gene_ID", "ks_stat", "gene_length"]
```

33. Load up a table containing a list of names for the genes affected by the differing conditions between our 2 datasets. The affected genes will be referred to as target genes or targets. A list of gene targets for our testing dataset can be found in the "Datasets/condition_targets" folder on our OSF project.

{Python3}

```
> target_names = pd.read_csv("<path/to/target_table.csv>", names = ["gene_ID"])
```

34. Merge the table of K-S statistics and the table of target gene names into a new table using Pandas' merge function. This new table will have an indicator column that shows if one of the target genes matched to one of the genes in the table of K-S statistics:

{Python3}

```
> temp_df = pd.merge(ks_table, target_names, how = "left", on
= "gene_ID", indicator = True)
```

35. Clean up the new table by removing any duplicates that may have been caused by multiple transcripts having the same gene name:

{Python3}

```
> temp_df.drop_duplicates(subset ="gene_ID",keep = "first", inplace = True)
```

36. Use the indicator column created in step 34 to create 2 subsets of our table of K-S statistics; one that only includes genes which matched with our target genes and one that only includes genes which did not match with our target genes:

{Python3}

```
> # Filter the temporary dataframe to only include targets
> targets = temp_df[temp_df._merge == "both"]
> # Filter the temporary dataframe to only include non-targets
> non_targets = temp_df[temp_df._merge == "left_only"]
```

37. Use matplotlib's scatter and violinplot functions to create plots which visualize the K-S statistic for all of the target and non-target genes (Figure 3A):

{Python3}

```
> # Initialize axes objects
> fig = figure(figsize = (5.5,4.5))
> ax = fig.add_axes([0,0,.9,.9])
> # Add violin plots
> violin_parts =
  ax.violinplot([non_targets.ks_stat,targets.ks_stat],
  showmeans = True)
# Alter violin plot colours
> for pc in violin_parts["bodies"]:
> pc.set_facecolor("grey")
> pc.set_edgecolor("grey")
> for partname in ("cbars","cmins","cmaxes","cmeans"):
> vp = violin_parts[partname]
> vp.set_edgecolor("grey")
```

```
> # Add dotplots
> x = np.random.normal(1, 0.04, size=len(non_targets.ks_stat))
> ax.scatter(x, non_targets.ks_stat, s = 12, color =
  "darkorange", alpha=0.8)
> x = np.random.normal(2, 0.04, size=len(targets.ks_stat))
> ax.scatter(x, targets.ks_stat, s = 12, color =
  "darkturquoise", alpha=0.8)
> # Determine x and y axis limits
> ax.set_xlim(0.5,2.5)
> ax.set_ylim(0,0.7)
> # Label the axes.
> positions = (1,2)
> labels = ("Non-targets", "Targets")
> ax.set_xticks(positions, labels, fontsize = 13)
> plt.ylabel("K-S Statistic", fontsize = 13)
> # Create grid lines
> axes = plt.gca()
> axes.yaxis.grid(linestyle = "-")
```

38. Divide the data into low, medium, and high K-S fractions and determine the fold enrichment of the target genes in each fraction using the determine_enrichment function from utilities.py. The K-S fractions in this example are set as genes with a K-S statistic less than 0.15, genes with a K-S statistic between 0.15 and 0.3, and genes with a K-S statistic above 0.3:

{Python3}

```
> # Define the cut-off for the high K-S fraction and the number of fractions to create.
> upper_ks = 0.3
> N_cats = 2
> enrich, sections = utils.determine_enrichment(targets, non_targets, upper_ks, N_cats)
```

Note: The 0.3 cut-off for the high K-S fraction was empirically determined based on numerical simulations of initiation limited and elongation limited translation (Flanagan et al., 2022). This value can be altered to increase or decrease the sensitivity of the method.

39. Perform Fisher's exact test to determine if the enrichment of targets in any of the K-S fractions is significant. The Fisher_exact_p_values function from utilities.py can be used to automatically calculate these P-values for each of the K-S fractions. This function automatically performs three tests, so the outputted P-values should be adjusted using the Benjamini Hochberg method. This can be done using statsmodels' multipletests function:

{Python3}

```
> # Perform Fisher's exact for all three fractions simultaneously.
> p_values = utils.Fisher_exact_p_values(targets, non_targets, sections)
> # Adjust the P-values using the Benjamini Hochberg method
> adj_p_values = multipletests(p_values, method = "fdr_bh")[1]
```

40. Use matplotlib's pyplot.bar function to create a series of barplots that show the fold enrichment of genes that are targets in each fraction (Figure 3A):

{Python3}

```
> # Specify the figure size.
> figure(figsize = (6.5,5.5))
> # Create a barplot showing enrichment in each fraction.
> bps = plt.bar([1,2,3],enrich, width = 0.5,
tick_label = ["Low K-S fraction","Medium K-S fraction",
"High K-S fraction"], color = ["g", "b", "m"], edgecolor =
"black")
> # Adjust fontsize and labels.
> plt.xticks(fontsize = 13)
> plt.ylabel("Fold Enrichment", fontsize = 13)
> # Increase plot margins
> plt.margins(0.1,0.1)
> # Create grid lines
> axes = plt.gca()
> axes.yaxis.grid(linestyle = "-")
> # Write the adjusted p-values on top of each bar.
> for b, p in zip(bps, adj_p_values):
  > height = b.get_height()
  > plt.annotate("p = " + "{}".format("%.2E" % Decimal(p)),
    xy=(b.get_x() + b.get_width() / 2, height),
    xytext=(0, 3), textcoords="offset points",
    ha="center", va="bottom")
```

Optional: After completing the analysis, the following optional steps can be used to create graphs showing the count arrays and smoothed cumulative read distributions for individual genes of interest in order to observe the changes in read distribution. To recreate the plots seen in Figures 4 or 5 using the testing data, "gene_of_interest" should be replaced with "Selenop" or "Aox3" respectively.

41. Choose a gene of interest and then find its count array for both the condition 1 and condition 2 datasets:

{Python3}

```
> # Select gene of interest
> goi = "gene_of_interest"
> # Initialize lists to hold the count arrays.
> goi_array_cond1 = []
> goi_array_cond2 = []
> # Iterate through the condition 1 and condition 2 datasets and extract count arrays from the
gene of interest.
> for tr_1, tr_2, name in zip(condition1, condition2,
  names_1):
  > if name == goi:
    > goi_array_cond1 = tr_1
    > goi_array_cond2 = tr_2
  > if len(goi_array_cond1) == 0:
    > raise ValueError("Gene name not found")
```

42. Create smoothed count arrays for the gene of interest:

{Python3}

```
> # Smoothed array from condition 1 dataset
> smoothed_array_1, cumul_1 =
utils.get_smoothed_array(goi_array_cond1 + 0.00000000001)
> # Smoothed array from condition 2 dataset
> smoothed_array_2, cumul_2 =
utils.get_smoothed_array(goi_array_cond2 + 0.00000000001)
```

43. Use Matplotlib's bar function to create bar plots that show the raw count arrays for the gene of interest:

{Python3}

```
> # Find the maximum read density between both arrays so it can be used to define the y-axis range.
> maxi = max([max(goi_array_cond1/sum(goi_array_cond1)),
max(goi_array_cond2/sum(goi_array_cond2))])
> # Create the bar plot for the condition 1 count array.
> plt.bar(list(range(len(goi_array_cond1))),
goi_array_cond1/sum(goi_array_cond1), width = 4)
> # Define the y-axis range
> plt.ylim([0,maxi*1.1])
```

```
> # Add the axis labels and title
> plt.ylabel("Read Density (normalized)", fontsize = 11)
> plt.xlabel("Transcript Position", fontsize = 11)
> plt.title("Condition 1" + goi, fontsize = 13)
> # Create grid lines
> axes = plt.gca()
> axes.yaxis.grid(linestyle = "-")
> # Display plot
> plt.show()
> # Create the bar plot for the condition 1 count array.
> plt.bar(list(range(len(goi_array_cond1))),
  goi_array_cond2/sum(goi_array_cond2), color = "darkorange",
  width = 4)
> # Define the y-axis range
> plt.ylim([0,maxi*1.1])
> # Add the axis labels and title
> plt.ylabel("Read Density (normalized)", fontsize = 11)
> plt.xlabel("Transcript Position", fontsize = 11)
> plt.title("Condition2 " + goi, fontsize = 13)
> # Create grid lines
> axes = plt.gca()
> axes.yaxis.grid(linestyle = "-")
```

44. Use Matplotlib's basic plot function to create line graphs that show the smoothed count arrays and the cumulative smoothed count arrays for the gene of interest:

{Python3}

```
> # Plot the smoothed count arrays from both datasets
> plt.plot(smoothed_array_1, label = "condition1")
> plt.plot(smoothed_array_2, label = "condition2", color =
"darkorange")
> # Add the axis labels, title, and legend
> plt.ylabel("Read Density", fontsize = 11)
> plt.xlabel("Transcript Position", fontsize = 11)
> plt.title("Smoothed and Normalized Count Arrays", fontsize =
13)
> plt.legend()
> # Create grid lines
```

```
> axes = plt.gca()
> axes.yaxis.grid(linestyle = "-")
> # Display plot
> plt.show()
> # Calculate the K-S statistic from the cumulative distributions.
> ks = max(abs(cumul_1 - cumul_2))
> # Plot the smoothed count arrays from both datasets
> plt.plot(cumul_1, label = "condition1")
> plt.plot(cumul_2, label = "condition2")
> # Write the K-S statistic on the plot
> plt.text(len(cumul_2)*0.66, 0.2, "KS stat = " +
str(round(ks,3)), fontsize = 11)
> # Add the axis labels, title, and legend
> plt.ylabel("Cumulative Read Density", fontsize = 11)
> plt.xlabel("Transcript Position", fontsize = 11)
> plt.title("Cumulative Distributions", fontsize = 13)
> plt.legend()
> # Create gridlines
> axes = plt.gca()
> axes.yaxis.grid(linestyle = "-")
```

EXPECTED OUTCOMES

Detecting changes in translation efficiency

This protocol will produce a list of genes whose translation efficiency is altered between two conditions. The output of this section is a list of the translationally altered genes, the magnitude of the change in translation efficiency, and the adjusted p-value indicating statistical significance.

The output of this section also includes three plots that visualize the results. Figure 2A uses a scatterplot to show the relationship between changes in translation efficiency and the number of collected read fragments via ribosome profiling. Figure 2B shows the distribution of changes in translation efficiency for all genes via a histogram. Figure 2C is a scatterplot which visualizes the deviation in the empirical dispersion between the ribosome profiling and RNA sequencing data. This deviation is used by RiboDiff to determine if dispersion should be estimated separately for the RNA sequencing and ribosome profiling datasets (Zhong et al., 2017).

Differential ribosome distribution analysis

This protocol will produce a list of K-S values on a per-gene basis, which measures the difference in the cumulative distribution associated with two ribosome footprint densities. Higher K-S values are found for genes whose distribution of ribosome footprints is substantially altered (Figure 4), whereas genes with low K-S values have similar ribosome distributions along their transcripts (Figure 5).

Figure 3 illustrates the results obtained when we apply our protocol to two experimental datasets. The first dataset (Figure 3A), includes data from liver cells collected from Trsp knockout mice and control mice. Trsp encodes for a tRNA which transports selenocysteine, a rare amino acid that is found in

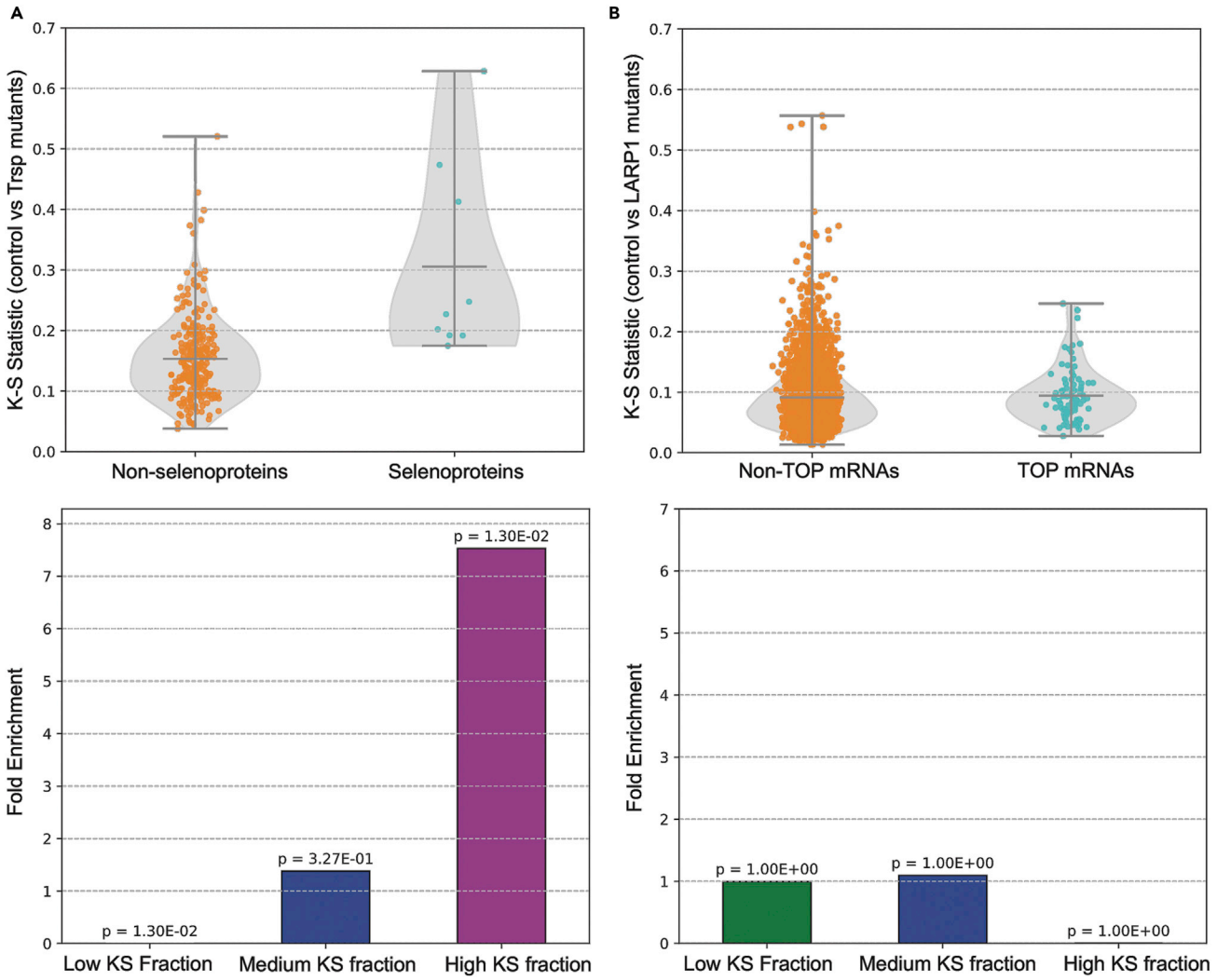


Figure 3. Differential analysis of ribosome profiling data for data treated to induce elongation limitation

Treatments included the depletion of selenocysteine carrying tRNA which arrested the elongation of selenoproteins (Fradejas-Villar et al., 2017), and the introduction of Torin 1 which inhibited the initiation of translation for transcripts that contain 5′TOP regions (Philippe et al., 2020). Dot plots with overlayed violin plots show the distribution of K-S statistics for genes which are targets and non-targets for the treatment. Bar plots show the enrichment of treatment targets in low, medium, and high K-S fractions. P-values were calculated using Fisher' exact test and adjusted using the Benjamini Hochberg method. Enrichment in the high K-S fraction and changes in K-S statistic distributions are only present for selenoproteins and non-selenoproteins (A), and not for TOP mRNAs and non-TOP mRNAs (B). This suggests that only limiting elongation rates impacts our K-S statistic metric.

selenoproteins. The depletion of this tRNA is expected to induce elongation limited translation for selenoproteins. In the second dataset (Figure 3B), a treatment with Torin 1 was applied to LARP1 knockout mutant and control human kidney cell cultures. Torin 1 significantly inhibits the translation of mRNAs with 5′ TOP motifs in wild type cells, but LARP1 mutant cells are resistant to this treatment. Therefore, we expect changes to the initiation rate of 5′ TOP containing mRNAs but no changes in elongation rate.

When comparing ribosome footprints in cells expressing or lacking selenocysteine carrying tRNA, we observe that mRNAs which encode selenoproteins have altered ribosome footprint distributions leading to relatively high K-S statistic values (Figure 3A). In contrast, ribosome footprints from cells that are sensitive or insensitive to initiation inhibiting treatment have generally low K-S statistic values for all genes (Figure 3B). Genes whose translation efficiency is altered, as determined by

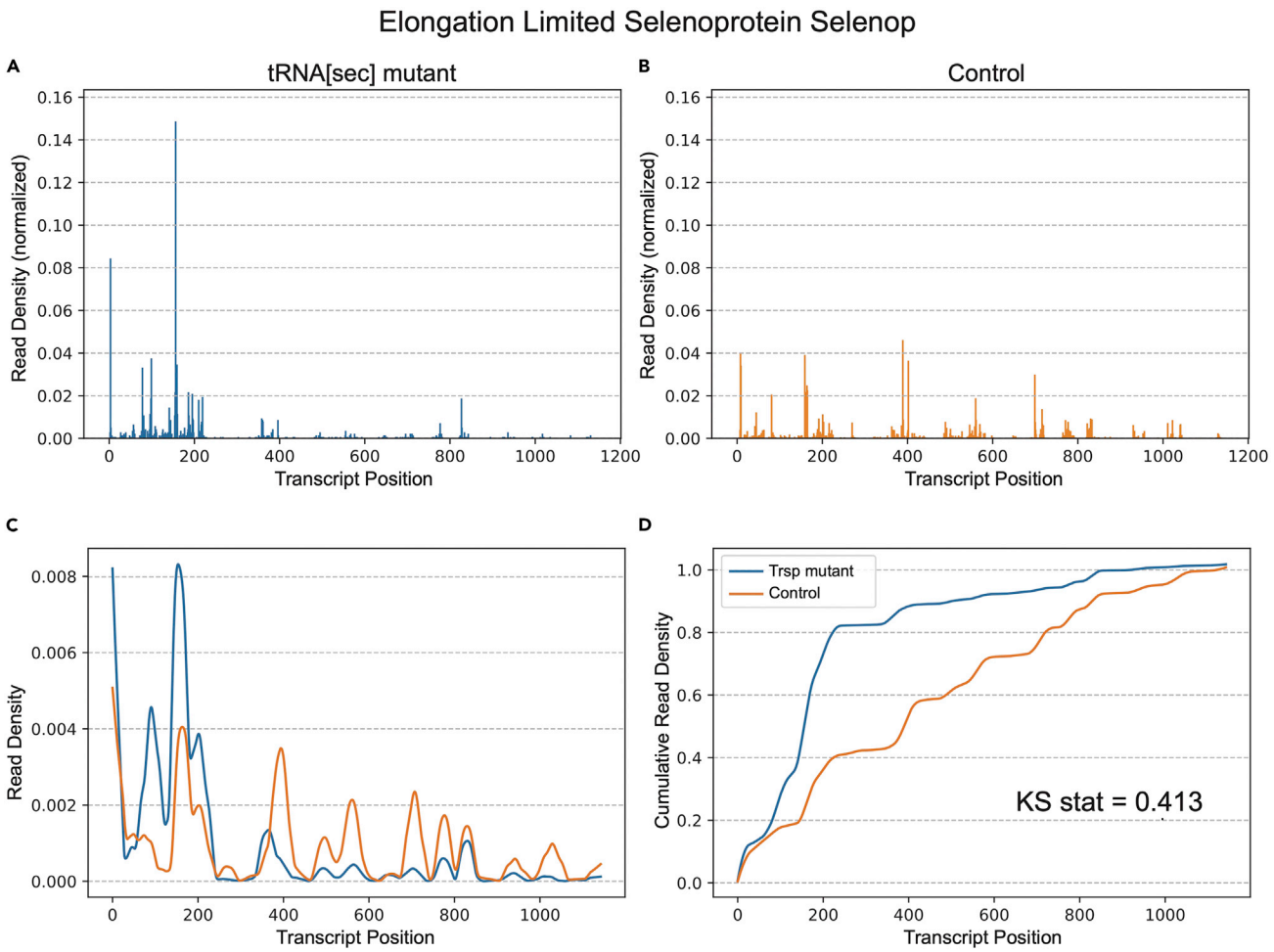


Figure 4. Determination of the K-S statistic for elongation limited selenoprotein Selenop

(A and B) Barplots depicting the normalized read density distribution for a gene from the condition 1 dataset (A) and the condition 2 dataset (B) (see also Figure 3A).

(C and D) Lineplots comparing the smoothed, normalized read densities (C) and the cumulative read densities (D) between condition 1 and 2. The K-S statistic is calculated as the maximum distance between the two cumulative distributions.

RiboDiff, can be analyzed individually to determine whether the change in translation efficiency is likely due to altered translation initiation or elongation.

If the treatment is sufficient to induce elongation limited translation in some or all of its targets, then it is expected that the targets will be greatly enriched within the subset of genes with high K-S statistics (Figure 3A). If the treatment does not cause elongation to become limiting, then no enrichment of targets should be observed (Figure 3B). The significance of this enrichment can be evaluated using Fisher's exact test.

LIMITATIONS

Given that this protocol utilizes a combination of open-source software packages and in-house code, each of the major sections have specific limitations. The computational requirements of the processing raw ribosome profiling data and RNA-sequence data section are relatively high when working with organisms with large genomes and may be a limiting factor for some users. The detecting changes in translation efficiency section require significant sequencing depth to yield statistically significant results. Similarly, low sequencing depth may lead to very few genes passing the filter in

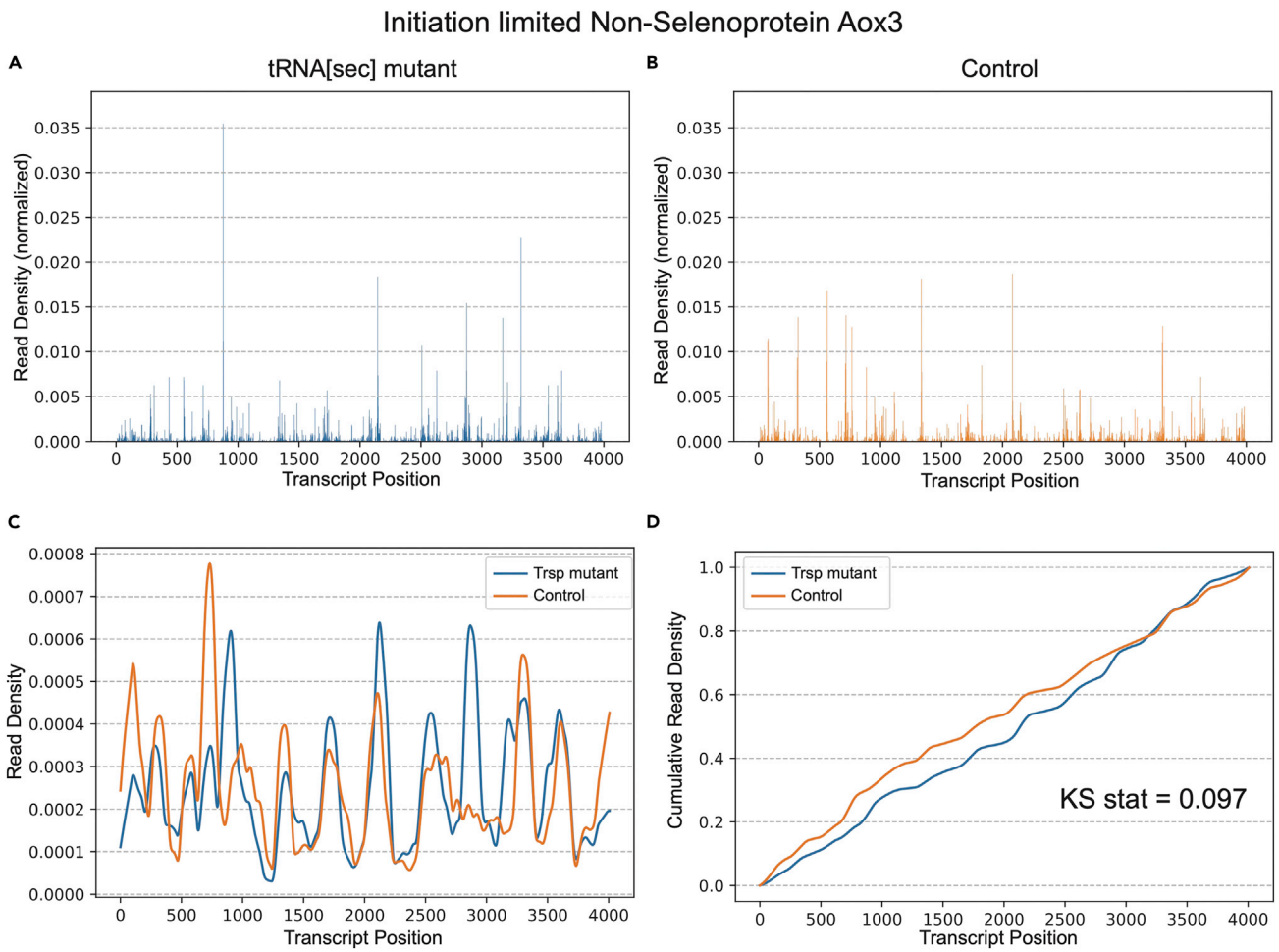


Figure 5. Determination of the K-S statistic for initiation limited non-selenoprotein Aox3 (see also Figure 3A)

(A and B) Barplots depicting the normalized read density distribution for a gene from the condition 1 dataset (A) and the condition 2 dataset (B).

(C and D) Lineplots comparing the smoothed, normalized read densities (C) and the cumulative read densities (D) between condition 1 and 2. The K-S statistic is calculated as the maximum distance between the two cumulative distributions.

step 26 which will be detrimental to the analysis, especially if the target genes are among those filtered out.

When a limiting elongation regime occurs, it is associated with a queue that starts from the position of the stalling site. Therefore, the selection from the KS metric may be biased toward detecting changes associated with targets located closer to the 3'end, with also less sensitivity to noise and sequencing depth in this case. Furthermore, our method is unable to differentiate between premature termination and stalling, as both events lead to a rapid drop-off in ribosome density after a certain point along a transcript. After this protocol is used to create a list of genes predicted to be elongation-limited, we suggest performing further experiments and analyses to confirm these predictions. For example, western blotting can be used to determine the size of a protein of interest and test for premature termination.

TROUBLESHOOTING

Problem 1

riboWaltz fails to install (before you begin step 16). riboWaltz requires several dependencies from CRAN and Bioconductor. If difficulties with the installation of riboWaltz occur, we highly recommend

looking over the installation instructions on the riboWaltz GitHub repository and/or submit an inquiry to their GitHub Issues page. If the installation error persists then we encourage users to open up an issue on this protocol's GitHub repository so that we can address the error directly.

Below we provide an example solution to a common problem where one of riboWaltz's dependencies fails to install due to a missing compression package called lzma.h on machines running Ubuntu.

Error message: " error: lzma.h: No such file or directory".

Potential solution

open up a terminal and then run:

```
> sudo apt -y install liblzma-dev
```

to install the lzma package.

Problem 2

Significant data loss occurs during pre-processing and alignment (steps 1–6).

Potential solution

- Check to make sure that the correct adapter sequence is being used during step 1.
- This analysis requires read data which has a Phred quality score of at least 33. If the data being used does not meet this quality threshold, then higher quality data will need to be acquired. Alternatively, the quality threshold can be reduced by altering the command in -q command in step 1 of the protocol, but this is not recommended.

Problem 3

STAR terminates prematurely when trying to generate the genome index with a message at the end of the output that says "Killed" (step 4). A common cause of this problem is insufficient available RAM. Below are several methods that can be used to either reduce the amount of RAM required by STAR or free up more RAM on a system.

Potential solution

- Reduce the number of threads specified by the argument "–runThreadN" to 1.
- Add the additional arguments "—genomeSAsparseD 2" and "—genomeSAindexNbaseat 12" to the STAR command in step 3.
- Close any other additional processes that are running on the machine.
- Check to make sure that no additional users are running RAM intensive processes on the machine via SSH or other remote connection.
- Upgrade to a machine with more RAM.

Problem 4

Upon loading in the GTF file as a dataframe, all of the values for the 5'and 3' UTR regions are zero (step 15). This problem is mostly likely caused by an annotation file that is missing information on the length of either the 5' and 3' UTR regions or the CDS region.

Potential solution

- Search for an alternative GTF file which has data on the 5', 3', and CDS regions for each gene.
- Rather than using riboWaltz to determine appropriate p-site offsets for each read length, a single offset value can be applied to every read. This can be done by replacing "utils.VariableThreePrimeMapFactory" with Plastid's "ThreePrimeMapFactory" function within step 23 of the protocol.

Problem 5

A key-error arises which says that "gene_name" cannot be found within one of the transcript objects (step 27). This occurs because the GTF file being used includes transcripts that do not map to any coding regions and therefore have no gene names.

Potential solution

Subset the list of transcript objects so that only protein coding transcripts are present. The following code is an example of this type of sub-setting:

```
{Python3}
> protein_coding = []
> For tra in transcripts:
  > if tra.attr["transcript_biotype"] == "protein_coding":
    > protein_coding.append(transcript)
> transcripts = protein_coding
```

RESOURCE AVAILABILITY

Lead contact

Further information and requests for resources and reagents should be directed to and will be fulfilled by the lead contact, Ethan Greenblatt, ethan.greenblatt@ubc.ca.

Materials availability

This study did not generate new unique reagents.

Data and code availability

The datasets generated during this study and the notebooks and additional code required to run this study are available on our OSF repository. The corresponding accession number is OSF: 5qcwk. Source data for the fastq files used in the paper is available at SRA: SRP078005. https://www-ncbi-nlm-nih-gov.ezproxy.u-pec.fr/Traces/study/?acc=GSE84112&o=acc_s%3Aa

ACKNOWLEDGMENTS

This work was supported by the Simons Foundation Autism Research Initiative (K.F. and E.J.G.), BioTalent Canada (K.F.), NSERC Discovery Grant (PG 22R34686), and the Peter Wall Institute of Advanced Studies (K.D.D.).

AUTHOR CONTRIBUTIONS

Writing, K.F., E.G., K.D.D., and W.L.; development, K.F., E.G., and K.D.D.; funding acquisition, E.G. and K.D.D.; Testing, K.D.D., W.L., and K.F.

DECLARATION OF INTERESTS

The authors declare no competing interests.

REFERENCES

Dobin, A., Davis, C.A., Schlesinger, F., Drenkow, J., Zaleski, C., Jha, S., Batut, P., Chaisson, M., and Gingeras, T.R. (2013). STAR: ultrafast universal RNA-seq aligner. Bioinformatics *29*, 15–21.

Dunn, J.G., and Weissman, J.S. (2016). Plastid: nucleotide-resolution analysis of next-generation sequencing and genomics data. BMC Genom. *17*, 958.

Erdmann-Pham, D.D., Son, W., Dao Duc, K., and Song, Y.S. (2021). EGGTART: a tool to visualize the dynamics of biophysical transport under the inhomogeneous l-TASEP. Biophys. J. *120*, 1309–1313.

Erdmann-Pham, D.D., Dao Duc, K., and Song, Y.S. (2020). The key parameters that govern translation efficiency. Cell Syst. *10*, 183–192.e6.

Flanagan, K., Baradaran-Heravi, A., Yin, Q., Dao Duc, K., Spradling, A.C., and Greenblatt, E.J. (2022). FMRP-dependent production of large dosage-sensitive proteins is highly conserved. Genetics, iyac094. https://doi.org/10.1093/genetics/iyac094.

Fradejas-Villar, N., Seeher, S., Anderson, C.B., Doengi, M., Carlson, B.A., Hatfield, D.L., Schweizer, U., and Howard, M.T. (2017). The RNA-binding protein Secisbp2 differentially modulates UGA codon reassignment and RNA decay. Nucleic Acids Res. *45*, 4094–4107.

Gordon, Assaf (2010). FASTQ/A short-reads pre-processing tools. http://hannonlab.cshl.edu/fastx_toolkit.

Harris, C.R., Millman, K.J., Van Der Walt, S.J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N.J., and Oliphant, T.E. (2020). Array programming with NumPy. Nature *585*, 357–362.

Howe, K.L., Achuthan, P., Allen, J., Allen, J., Alvarez-Jarreta, J., Amode, M.R., Armean, I.M., Azov, A.G., Bennett, R., Bhai, J., and Flicek, P. (2021). Ensembl 2021. Nucleic Acids Res. *49*, D884–D891.

Hunter, J.D. (2007). Matplotlib: a 2D graphics environment. Comput. Sci. Eng. *9*, 90–95.

Ingolia, N.T., Brar, G.A., Rouskin, S., McGeachy, A.M., and Weissman, J.S. (2012). The ribosome profiling strategy for monitoring translation in vivo by deep sequencing of ribosome-protected mRNA fragments. Nat. Protoc. *7*, 1534–1550.

Kent, W.J., Sugnet, C.W., Furey, T.S., Roskin, K.M., Pringle, T.H., Zahler, A.M., and Haussler, D. (2002). The human genome browser at UCSC. Genome Res. *12*, 996–1006.

Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B.E., Bussonnier, M., Frederic, J., and Willing, C. (2016). Jupyter Notebooks-A Publishing Format for Reproducible Computational Workflows (Elpub), pp. 87–90.

Langmead, B., and Salzberg, S.L. (2012). Fast gapped-read alignment with Bowtie 2. Nat. Methods *9*, 357–359.

Lauria, F., Tebaldi, T., Bernabò, P., Groen, E.J.N., Gillingwater, T.H., and Viero, G. (2018). ribowaltz: optimization of ribosome p-site positioning in ribosome profiling data. PLoS Comput. Biol. *14*, e1006169.

Li, B., and Dewey, C.N. (2011). RSEM: accurate transcript quantification from RNA-Seq data with or without a reference genome. BMC Bioinf. *12*, 1–16.

Liao, Y., Smyth, G.K., and Shi, W. (2014). featureCounts: an efficient general purpose program for assigning sequence reads to genomic features. Bioinformatics *30*, 923–930.

Philippe, L., van den Elzen, A.M.G., Watson, M.J., and Thoreen, C.C. (2020). Global analysis of LARP1 translation targets reveals tunable and dynamic features of 5′ TOP motifs. Proc. Natl. Acad. Sci. USA *117*, 5319–5328.

Reback, J., McKinney, W., van den Bossche, J., Augspurger, T., Cloud, P., Klein, A., Hawkins, S., Roeschke, M., Tratner, J., She, C., et al. (2022). pandas-dev/pandas: Pandas 1.3.4 (zenodo). https://doi.org/10.5281/zenodo.5574486.

Santesmasses, D., Mariotti, M., and Gladyshev, V.N. (2020). Tolerance to selenoprotein loss differs between human and mouse. Mol. Biol. Evol. *37*, 341–354.

Van Rossum, G., and Drake, F.L. (2009). Python 3 Reference Manual (CreateSpace).

Wickham, H., Hester, J., Chang, W., and Bryan, J. (2022). devtools: Tools to Make Developing R Packages Easier. https://devtools.r-lib.org/.

Woolstenhulme, C.J., Guydosh, N.R., Green, R., and Buskirk, A.R. (2015). High-precision analysis of translational pausing by ribosome profiling in bacteria lacking EFP. Cell Rep. *11*, 13–21.

Zhong, Y., Karaletsos, T., Drewe, P., Sreedharan, V.T., Kuo, D., Singh, K., Wendel, H.G., and Rätsch, G. (2017). RiboDiff: detecting changes of mRNA translation efficiency from ribosome footprints. Bioinformatics *33*, 139–141.