StructureFold Manual
revision 1.0

StructureFold is a suite of Python scripts to efficiently process and analyze data from DMS(-) and DMS(+) libraries generated by the StructureSeq technique. It is divided into several modules, some of which have other software dependencies. We have attempted to give the user as much mod ability as possible in all data formats, as to better facilitate working with any possible experimental design.

Dependencies

► Python 2.7.1+
All scripts are written in Python, making this a needed dependency.

●cutadapt(https://cutadapt.readthedocs.io/en/stable/)
Used by: trim_fastqs.py
cutadapt is an excellent Python package for removing adapters from reads. It can be used independently of the batch script, but it exists for your conveninece.

► Bowtie2 (http://bowtie-bio.sourceforge.net/bowtie2/index.shtml)
Used_by:bowtie2_batcher.py
Bowtie2 is the recommended mapping program to map RNAseq reads onto transcripts. It is definitely possible to use any of several other aligners so long as they take input .fastq and output into the SAM/BAM format.Bowtie2 can be run independently of the batch script which is provided as a convenience.

► SAMtools (http://samtools.sourceforge.net/)
Used_by: process_sams.py
Samtools is required for dealing with SAM/BAM format files. Samtools can be run independently of the batch script, which is provided as a convenience.

► RNAStructure package (http://rna.urmc.rochester.edu/RNAstructure.html)
Used by: batch_fold.py
An RNA folding program able to use the .react restraint files generated by previous steps. RNAStructure or the Vienna Package is needed for batch_fold.py to batch fold all given RNA structures.

► Vienna Package (https://www.tbi.univie.ac.at/RNA/)
Used by: batch_fold.py
An RNA folding program unable to use the .react restraint files generated by previous steps. RNAStructure or the Vienna Package is needed for batch_fold.py to batch fold all given RNA structures.

► FastQC (http://www.bioinformatics.babraham.ac.uk/projects/fastqc/)
We highly recommended that all input .fastq files are first inspected for obvious indescrepancies and or issues that resulted from sequencing.

►R

We highly recommended the use of R for working with all .csv files that are the final data format StructureFold produces.

Instructions

▼Step 1 – Data Inspection

There really isn't all that much to say here. Run your .fastq files through FastQC and check that yield, quality, etc., are not greatly different than you would expect from the sequencing runs.

▼Step 2 - Data filtering I

The purpose of this step is to remove any adapter contamination and filter out reads which are too short, too long, or of questionable quality, before mapping to the transcriptome.

●Method 1

Run trim_fastq.py in the directory with all of your .fastq files. It will automatically run a suite of commands using cutadapt to remove the forward adapter twice and the reverse adapter once, as well as filter reads by quality and length. The script will automatically rename all output files, and delete all intermediate files. Use the -h flag to see the complete options when running this script.

●Method 2

Run the following cutadapt steps manually on all .fastq files:
cutadapt -n 2 -g [fiveprime adapter sequence]-o [out1] [input_file]
cutadapt -a [threeprime adapter sequence] -o [out2] [outputfile1]
cutadapt -m [min read length] -g [min read quality] -M [max read length] -o [out3] [out2]

▼Step 3 – Read Mapping

This step is very straight forward – the filtered reads from the previous step are to be mapped against all of the transcripts present in your study subject.

●Method 1

Run bowtie2_batcher.py in the directory with all of your filtered .fastq files after updating the path to your bowtie2 genome index. By default, this script tells bowtie2 to use 8 cores, but this is also configurable in the script itself.

●Method 2

Run bowtie2 on every trimmed .fastq file, with the following command as a guideline:
bowtie2 -a -p 8 -x m_index -q [infile] > [outfile]

▼Step 4 – Data filtering II

This step aims to deal with mappings which either have too many mismatches (more than 3) or have a mismatch on the first base pair of the mapping , which theoretically should not be allowed to happen with the reactivity protocol. Additionally, only reads correctly mapping to the appropriate strand for the transcript are kept.

●Method 1
Run process_sams.py
This will automatically go through every .sam in the directory and make the appropriate changes, writing all new files with suffixes.

●Method 2
Run SAMtools on your output SAM once to only keep forward mapping reads
samtools view -h -F 0xeff -S [infile] > [outfile]
Figure out another way to filter out reads which have > 3 mismatches, and no mismatch on the first base pair.

If you you wish to combine data sets, this is one point where you can do so. Simply by using the terminal command 'cat', combine SAM files before the next step.

▼Step 5 – Generate RT stops.

This step will use your transcript file and the filtered SAM files to generate RT stop count files. This will greatly compact the data in terms of size. There is no other way to do this part of the protocol besides the provided Python script.

●Method 1
Run get_RT_stop.py
This will generate .rtsc files

This is another step where data can be combined, using the combine_RT_stop.py script.
Coverages may be computed with both get_coverage.py
view_coverages.py and generate_coverages.py will calculate overlaps, and generate overlap files which are useful later in the pipleline.
Specificity may be computed with get_specificity.py

▼Step 6 – Calculate Reactivity

This step uses stop files <.rtsc> from the previous step to generate reactivty files <.react>. It requires use of both (-)DMS .rtsc and (+)DMS .rtsc files. A full help menu is available with use of the -h flag.

●Run reactivity_calculator.py

This represents a fork in the pipleline. Once you have the .react files, you are free to use them in one of three ways. They may be analyzed as they are (GREEN PATH), folded normally (BLUE PATH), or folded using the partition function option (RED PATH).

▼ Step 7.1 – Extract basic stats (GREEN PATH)

This is the most basic and straight forward analysis that can be done, and is the least computationally intensive.

7.1.1 Run reset_n_bp.py on your .react files. This will generate new .react files where the last n base pairs have been reset to 'NA', where n is a threshold you have determined based on your library generating protocol, 20bp by default.

It is possible to break apart the .react at this time into different pieces based on transcript annoation – i.e. break the transcript into 5'UTR, CDS, and 3' UTR and analyze each separately.

7.1.2 Run reactivity_stats.py with an appropriate coverage overlap file. The coverage file should be the overlap of coverages of at least n coverage between all (+)DMS samples used to generate the .react files in the current directory. The script will then take all .react files and turn them into a .csv with max reactivity, average reactivity, standard deviation of reactivity, and gini of reactivity.

Example:

Tissue_1_control.react
Tissue_1_treatment1.react
Tissue_1_controlxTissue_1_treatment1.txt

In a directory would yield stats_out.csv, containing data entries for all transcripts in the .txt coverage/overlap file from both samples.

Coverage overlap should be the intersection of both (+)DMS libraries or pools that went into the .react files. If sample A was (-)DMS control, and sample B was (+)DMS control, and C and D were a treatment respectively (-/+) DMS, than the overlap to use between the two would be BxD.txt. I.E.:

A.rtsc (-)DMS + B.rtsc (+)DMS → Tissue_1_control.react
C.rtsc (-)DMS + D.rtsc (+)DMS → Tissue_1_treatment1.react
overlap of (+)DMS libraries → BxD.txt

This resultant .csv can be directly loaded into R and analyzed.

▼ Step 7.2 – Folding (BLUE PATH)

This represents an intermediate in terms of computational requirements and analysis requirements.

As with the GREEN PATH, one needs to have a list of the RNAs to predict, which should be the overlap of coverage beyond n in both (+) DMS .rtsc files between the conditions you will compare. I.E.:

A.rtsc (-)DMS + B.rtsc (+)DMS → Tissue_1_control.react
C.rtsc (-)DMS + D.rtsc (+)DMS → Tissue_1_treatment1.react
overlap of (+)DMS libraries → BxD.txt

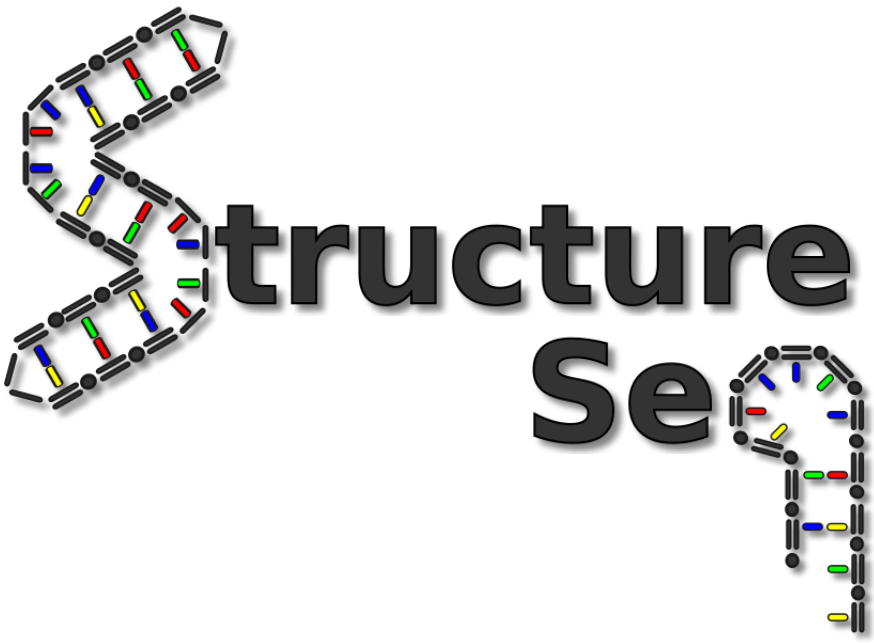
One would use BxD.txt

Run batch_fold.py without the partition function option, with the -mfe flag, and with the -sht x flag, where x is generally 20 – ignore the reactivities of the last 20 base pairs, on both control and treatment conditions. This will give you two directories of .CT files, which the next script operates on. Run get_PPV_file.py on these two directories and it will produce a .csv with the PPV of the transcripts between conditions. This .csv can be directly analyzed in R. Additionally, .ps visual models are also produced for easy visual representation of presented structures.

Hint: -T 295.15 is room temperature.

▼ Step 7.3 – Partition Folding (RED PATH)

*batch script included to automate this step



RMSD <.csv>

BP Files <.bp>

batch_get_rmsd.py

Raw reads <.fastq>

Cutadapt*

Filtered reads <.fastq>

Bowtie2*

Mapped reads <.sam>

get_RT_stop.py

RT Stops <.rtsc>

reactivity_ calculator.py

Reactivity <.react>

Base-pair probability matrix <.ps>

Reactivity stats <.csv>

batch_folder.py

Reactivity <.react>

reactivity_stats.py

CT Files <.ct>

generate_ PPV_file.py

Structure models <.ps>

PPV <.csv>