StructureFold 2.0 Manual
revision 1.0

Contributors
Yin Tang
David C. Tack

Purpose

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. The final analysis can be determined by the experimenter, as the suite's goal is to facilitate translating illumina reads into RNA structures quickly and efficiently, such that genome-wide analyses on structure (structureome) are approachable and practical for any experimenter.

Dependencies

►Python 2.7.1+

All modules are written in Python. While Python is not as computationally efficient as other methods, each step where it is used represents an intermediate step for which no other software exists to tackle the problem, making it useful in a data processing pipeline.

●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 convenience.

►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] files as restraint files generated by previous steps. RNAStructure or the Vienna Package is needed for batch_fold.py to batch fold all given RNA structures for which there is sequence data for.

►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 indiscrepancies 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. While not a hard dependency, R is simply too good to not use.

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, and duplication, are not greatly different than you would expect from the sequencing runs, and that no obvious artifacts are present in the data. Your results here may indicate that more extensive trimming and filtering are necessary in step 2, but this is something that needs to be diagnosed on a case by case basis.

●Method 1

Run FastQC on all [.fastq] files.

▼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, thus ensuring a more accurate calculation of reactivity and structure.

●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]
Where out3 is the final output of this step.

▼Step 3 – Read Mapping

This step is very straight forward, where 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 [your_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 given 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, and removing all intermediate files.

●Method 2

Run SAMtools on your output [.sam] once to only keep forward mapping reads:
samtools view -h -F 0xeff -S [infile] > [outfile]
Use another way to filter out reads with >3 mismatches, and no mismatch on the first base pair. Picard tools may have functionality for this.

If you you wish to combine data sets/samples/replicates, using the terminal command 'cat' between the respective [.sam] files is appropriate here, though they may be combined in the next step as well.

▼Step 5 – Generate RT stops.

This step will use your transcript [.fasta] 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 from [.sam] files

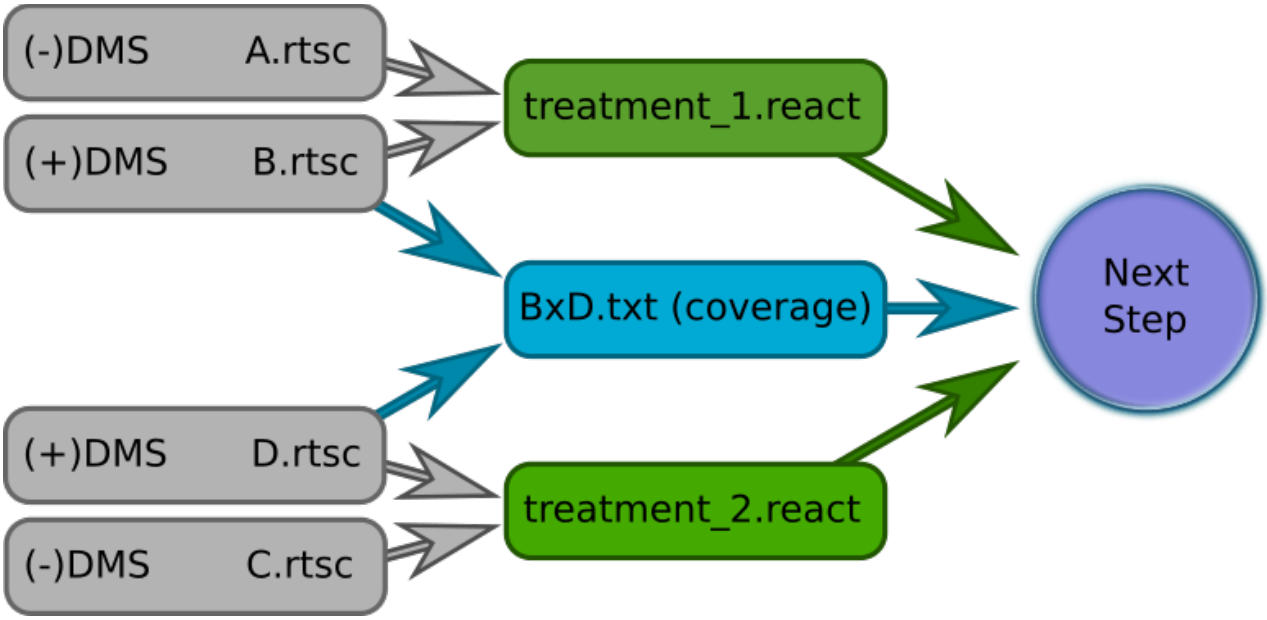
●combine_RT_stop.py may be used to combine [.rtsc] files

●Stop specificity may be computed with get_specificity.py

●Coverages must be computed with get_coverage.py
view_coverages.py and generate_coverages.py will generate files which are the overlapping transcripts with coverage greater than n (default =1) between all pairwise combinations of [.rtsc] files. For any comparison between conditions in future steps, the corresponding (+)DMS samples or combined samples between each condition or treatment need to be compared against each other. Transcripts which have coverage exceeding the threshold in both (+)DMS conditions between samples are transcripts which can have their structures and reactivities accurately compared and resolved, i.e.:

A.rtsc (-)DMS + B.rtsc (+)DMS → control.react
C.rtsc (-)DMS + D.rtsc (+)DMS → treatment_1.react
E.rtsc (-)DMS + F.rtsc (+)DMS → treatment_2.react
Overlap of (+)DMS libraries for control vs treatment_1→ BxD.txt
Overlap of (+)DMS libraries for control vs treatment_1→ BxF.txt
Overlap of (+)DMS libraries for treatment_1 vs treatment_2→ DxF.txt

When using [.react] files in subsequent steps, the coverage overlap list between the two constituent (+)DMS samples that went into the [.react] files represents the transcripts where a comparison is valid between the samples, but can only be calculated from raw coverage.



There are three possible next steps, each of which require this sort of data set up, namely two [.react] files, and the coverage overlap list between their constituent (+)DMS libraries.

▼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

●Make sure you have coverages for the (+)DMS overlap between each set of [.react] files.

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).

Each path requires the use of an overlap file, generated off of the coverage between the (+)DMS files used to make any two [.react] files. A transcript must have adequate coverage in both (+)DMS treatment conditions in order to be analyzed between conditions accurately.

▼Step 7.1 – Extract basic stats (GREEN PATH)

This is the most basic and straight forward analysis, and is the least computationally intensive.

● 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 annotation – i.e. break the transcript into 5'UTR, CDS, and 3' UTR and analyze each separately.

●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 in a directory and turn them into a [.csv] with max, average, and standard deviation of reactivity, as well as gini of reactivity, so that directory should only contain [.react] files corresponding to the coverage file, i.e, in a directory containing…

control.react
treatment1.react
BxD.txt overlap

run reactivity_stats.py 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 a given threshold in both constituent (+) DMS .rtsc files used to make the .react files between the conditions top be compared.

●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 two conditions ([.react files]) This will give you two directories of [.ct] files, which the next script operates on.

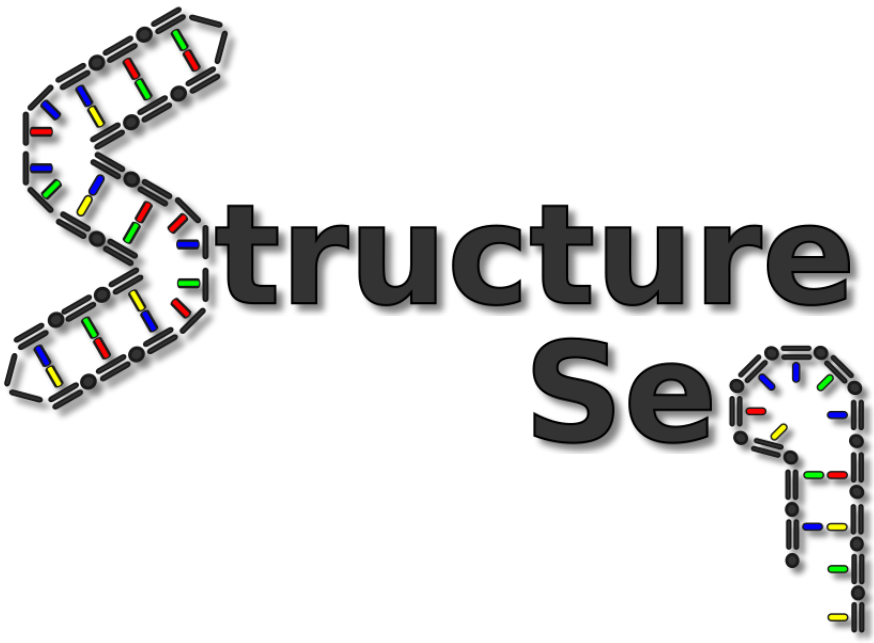
●Run get_PPV_file.py on these two directories and it will produce a [.csv] with the PPV of the transcripts between conditions. This [.csv] can be directly analyzed in R. Additionally, [.ps] visual models are also produced for easy visual representation of presented structures.

Hint: -T 295.15 is room temperature.

▼ Step 7.3 – Folding using partition function (RED PATH)

This represents the highest computationally demanding analysis in terms of time and processing. It requuires the same inputs of a list of transcripts, and [.react] files.

*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>