StructureFold2 Manual

Tools to facilitate rapid analysis of genome-wide chemical probing data on RNA structure

David C. Tack, Yin Tang, Laura E. Ritchey, Sarah M. Assmann, Philip C. Bevilacqua

Dependencies (Requirements)	Link
Python 2.7.X	https://www.python.org/
BioPython	http://biopython.org/
Numpy	http://www.numpy.org/
Cutadapt	http://cutadapt.readthedocs.io/en/stable/index.html
SAMtools	http://samtools.sourceforge.net/
Bowtie2	http://bowtie-bio.sourceforge.net/bowtie2/index.shtml

Recommended	Link
FastQC	https://www.bioinformatics.babraham.ac.uk/projects/fastqc/
R statistical language	https://www.r-project.org/
RNAstructure Package	https://rna.urmc.rochester.edu/RNAstructure.html
MEME Suite	http://meme-suite.org/
Vienna Package	https://www.tbi.univie.ac.at/RNA/

Introduction

StructureFold2 is a set of Python [1] scripts designed to efficiently and accurately process Structure-seq [2-4] or other high-throughput libraries generated to yield reverse transcription stops via chemically probing RNA structure into genome-wide transcript reactivity values at the single nucleotide resolution level (structurome). These reactivity values may be used as folding restraints for RNAstructure or similar software, greatly enhancing structure prediction with *in vivo*, genome-wide probing data. This package allows a user with a minimal computational or statistical background to execute a number of mostly automated processes that assemble the *in vivo* reverse transcriptase stops collected by any chemical probing method that blocks the processivity of reverse transcriptase, deriving values indicative of the probability of base pair single-strandedness from which a reasonable facsimile of the actual *in vivo* RNA structurome can be obtained. Ultimately, the final design and analysis of the data is up to the experimenter; data may be piped into RNAstructure [5]/ Vienna packages [6] to obtain predicted folds of transcripts, or the experimenter may choose to analyze the derived DMS reactivity or raw RT stop values in any way that suits their experimental design with typical statistical packages, such as R [7]. StructureFold2 is amenable to a variety of chemical probes (e.g. DMS, glyoxal, SHAPE) provided as there is a negative reagent control library included.

Before you begin…

An accurate StructureFold2 analysis hinges on a prudent selection of the transcriptome of interest. We highly recommend that the user take the time to carefully consider the version, quality of annotation, and size of the transcriptome to be used; the <.fasta> you initially choose that contains this information may not be changed during the course of an analysis. Some Eukaryotes contain an excessive number of transcript isoforms per gene, which may complicate downstream analyses, and it may be wise, e.g. to prune to no more than two isoforms per gene to map against. Selecting the most abundant isoform based on other techniques or experiments is an alternative approach to reduce the number of included transcripts per gene and enhance the precision of reactivity values and the simplicity of all downstream analyses. Ensembl [8] is a great place to look for transcriptome (cDNA) files if one has not already been selected to map against, although absolutely any <.fasta> file containing transcripts will work provided the transcripts are of sufficient length.

Website	Link
Ensemble	https://www.ensembl.org/info/data/ftp/index.html
Ensemble Plants	http://plants.ensembl.org/info/website/ftp/index.html
Ensemble Protists	http://protists.ensembl.org/info/website/ftp/index.html
Ensemble Bacteria	http://bacteria.ensembl.org/info/website/ftp/index.html

We recommend mapping to the whole transcripts before partitioning the resulting derived information into transcript features (e.g. 5'UTR, CDS, 3'UTR) in later steps so that reads overlapping two of these features will not encounter trouble mapping. To ensure that transcript annotation is sufficient to later subdivide features for individual analysis, check to ensure the start and stop coordinates of each of these features are included with the annotation or an associated <.gtf> or <.gff> file. At this time we are unable to provide an automated script to handle annotation and automatically perform this subdividison due to the lack of a consistent annotation file standard.

StructureFold2 is not a computationally demanding package by itself. Bowtie 2, the recommended short read aligner, is likewise lightweight and not exceptionally demanding on most hardware, even on larger transcriptomes and data sets. During the course of data processing however, many intermediate files will be generated; thus, having at least 4 TB of space available is advised to store these files until an analysis is completed. Using RNAStructure or Vienna Package to interpret the restraints generated by StructureFold2 with the included scripts to batch-fold thousands of transcripts is computationally demanding; this can tie up multiple cores for several days. Thus, access to a server or a dedicated personal machine is advised if carrying out this particular task. We recommend running StructureFold2 on either MAC OS or any mainstream Linux system. It is possible to run StructureFold2 on a Windows install within an emulated environment, but all features may not work properly and no additional support can be given.

Installation and Support

Download the entire package from Github and place in a directory of your choosing, or you may use git directly to clone the repository from Github with the command:

git clone https://github.com/StructureFold2/StructureFold2

Next, set the path to the folder containing the scripts, such that they may be called from any directory. This may be done by adding the following to your shell profile:

PATH=$PATH:/home/path/to/new/folder/StructureFold2

File permissions to the installation folder must now be set. Typically this will entail running the following command on the StructureFold2 folder which will grant all users read and execute permissions and all permissions to the owner:

chmod -R 755 StructureFold2/

Make sure to check the dependencies (requirements) of StructureFold2 and install these according to any instructions provided on respective websites and documentation. Many of these packages are commonly available from package managers or Python package managers native to either Linux or MacOS systems. For a listing of these dependencies, check the top of page 1. Depending how these are installed, permissions and the path may need to be set up for each dependency as StructureFold2 will automatically call some of these programs. To check that you have successfully installed StructureFold2 and all of the core dependencies, try executing the test script, test_sf2.py, in a new shell. The script will notify the user of any missing dependencies that are not available in the path or for which the user does not have adequate permissions to run. If you plan to customize the

StructureFold2 pipeline and forgo the use of cutadapt for read trimming and bowtie2 for read mapping, these dependencies do not need to be installed and set up.

   This manual makes the best attempt possible to detail the present StructureFold2 tool set and demonstrate its use. However, it is nearly impossible to plan for all contingencies. While the software is provided 'as is', we are eager to improve and further streamline our analysis pipeline. If you encounter a bug or require support, or have an idea for a feature that would improve your overall StructureFold2 experience, please include StructureFold2 in the subject line when writing to David Tack (kujiratan@gmail.com).

Data and file types

File Type, Extension		Contains	Input for Script Number:
FASTQ	<.fastq>	Illumina reads	1,2
FASTA	<.fasta>	Nucleotide sequences	2,4,5,A,H,J,K,L,R
SAM	<.sam>	Aligned sequences	3,4
RTSC	<.rtsc>	Reverse Transcriptase stops	5,A,I,J,K,L,M
SCALE	<.scale>	Reactivity Normalization Scale	5,A
REACT	<.react>	Reactivity values	E,F,G,H,O,P,Q
CT	<.ct>	Connectivity Data	B,C,D
DBN	<.dbn>	Dot Bracket Notation	
CSV	<.csv>	Any data, delimited by comas	

Contents

Core Scripts	#	Function
fastq_trimmer.py	1	Trims <.fastq> via Cutadapt
fastq_mapper.py	2	Maps <.fastq> into <.sam> via Bowtie 2
sam_filter.py	3	Filters <.sam> with samtools
sam_to_rtsc.py	4	Counts RT stops in <.sam> files
rtsc_to_react.py	5	Takes <.rtsc> files, calculates nucleotide reactivity

Auxiliary Scripts	Letter	Function
batch_fold_rna.py	A	Drives RNAStructure or ViennaPackage
make_stranded_csv.py	B	Calculates single-strandedness of folded RNAs
make_PPV_csv.py	C	Calculates PPV between directories of structures
make_MFE_csv.py	D	Organizes the MFE of folded RNAs.
react_statistics.py	E	Creates an easy to analyze <.csv> from <.react>(s)
react_RMSD.py	F	Calculates RMSD between two <.react>s
react_maxima.py	G	Finds shared overlap of maxima between <.react>s
react_windows.py	H	Finds windows of changed reactivity in <.react>s
rtsc_combine.py	I	Combines <.rtsc> files
rtsc_coverage.py	J	Calculates coverage per transcript on <.rtsc>
rtsc_correlation.py	K	Generates a <.csv> to examine correlation
rtsc_specificity.py	L	Generates a <.csv> to examine stop specificity
rtsc_abundances.py	M	Calculates transcript abundance from <.rtsc>
coverage_overlap.py	N	Creates lists of transcripts above a given coverage
react_combine.py	O	Combines <.react> files
react_bins.py	P	Bins reactivity values of transcripts
react_delta.py	Q	Calculates change between two <.react>s
fasta_composition.py	R	Logs sequence composition info into a <.csv>
test_sf2.py	S	Checks to see if dependencies are installed

Procedure

StructureFold2 is subdivided into two main analysis sections. The first section (Figure 1), uses the core scripts, takes the raw data and processes it into <.react> files, and can be thought of as a linear data processing pipeline. The second section, which uses auxiliary scripts, is more user-directed (Figure 2), and allows for any number of the modules to be run, each yielding a specific set of metrics generated or extracted from <.react> files. The Blue Path analyzes reactivity patterns directly, while the Green Path emphasizes metrics from the predicted folds of transcripts. During the course of the first section, we recommend using the logging options at each step and carefully reviewing the details of the respective trimming, mapping, and filtering taking palace. These useful diagnostics allow for the early detection of potential problems and the mitigation of downstream errors. However, the nuanced interpretation of results in light of both the questions and the structurome being analyzed at each step precludes automating the entire analysis process, thus we encourage the user to check the results of each step before proceeding.

◄0► Library quality control

These diagnostics are optional. The purpose of this step is to assess the quality of the user's structure-probing libraries before proceeding with further analyses, and to assure data persistence.

Validate and Archive libraries

Generate MD5 or SHA256 sums for all of your compressed sequence files and log these to a file stored alongside the files; this will enable verification of the original data at a later date or after a large transfer. These hashes are unique 'fingerprints' that remain constant so long as the contents of the file remain perfectly intact. It is also prudent to keep at least two parallel copies of all compressed sequence files on two separate devices, such that a device failure will not result in data loss, as the time, money, and effort invested to generate the libraries and sequences greatly exceeds the cost of a modest external or internal drive.

Check library quality

Run FastQC on your <.fastq> files. This will yield many useful diagnostics for your data. Things to look out for are a preponderance of low-quality scores (under 30 Phred) or repetitive artifacts, or excessive adapter content. Most of these issues will be sorted out by the next step, but it is up to the experimenter to determine if the libraries need remaking or re-sequencing if something looks catastrophic.

◄1► Read adapter and quality trimming

This step prepares reads to be mapped to a reference transcriptome by removing adapter and adapter fragment sequences from the reads, removing low-quality bases from the 3' end, and removing reads that are too short. For the greatest consistency and replication, we highly recommend using our script to run Cutadapt [9] on your files (1.A) using the recommended settings, although other read-trimming packages could be substituted in at this step (1.B).

1.A Run batch trimming script (recommended)

Locate your folder where all relevant unzipped <.fastq> files are located. Run the script fastq_trimmer.py. This will automatically remove 5' and 3' adapters, as well as remove low quality bases from the 3' end, and remove reads that have become or were too short, based on a user-defined threshold. Filtered files carry the same base name as the original files but are suffixed with '_trimmed' and saved to the same directory. Create a new folder for these trimmed <.fastq>s or store the raw <.fastq> files in another folder to prepare for the next

step. Additional options are available in the help menu of the script, which can be accessed with the '-h' option, including the option to use alternate adapters or change trimming thresholds.

① 🐍 fastq_trimmer.py

A.fastq → A_trimmed.fastq

fastq_trimmer.py drives three commands to Cutadapt: remove the 5' adapter up to twice, remove the 3' adapter once, remove low-quality (<30 Phred) bases from the 3' end while ensuring the finished read length is > 20 nt, with the option of removing reads over a certain length. The default settings should accommodate most experiments, and the default primers are those used in the Structure-seq2 protocol [4]. If you wish to retain a detailed log of what Cutadapt did, use the -log option; the default log name is trim_log.txt, though this is also modifiable (-logname). The rationale for an option to set a maximum allowed read length is to remove reads where no adapters have been trimmed, suggesting ambiguous read origin. Illumina's NextSeq and NovaSeq will use a slightly different quality score system; thus, use the -nextseq flag if your sequencing was done with the new sequencing chemistry.

Options
-h, --help	show this help message and exit
-log	Create an explicit log of the trimming
-nextseq	Use NextSeq/NovaSeq quality scores
-fp FP_ADAPT	[default = TGAACAGCGACTAGGCTCTTCA] 5' adapter
-tp TP_ADAPT	[default = GATCGGAAGAGCACACGTCTG] 3' adapter
-minlen MIN_LEN	[default = 20] minimum accepted sequence length
-minqual MIN_QUAL	[default = 30] minumum accepted base quality
-maxlen MAX_LEN	[default = None] maximum seq length
-suffix SUFFIX	[default = trimmed] trimmed <.fastq> file suffix
-logname LOGNAME	[default = trim_log] Name of the log file

1.B Custom read trimming (not recommended)

Run any software you choose to trim and filter the reads. All trimmed files should be placed in a new directory for the next step, or otherwise organized for the next part of the data processing pipeline. Structure-Seq libraries almost always use the default linker (see Structure-seq2 protocol [4]), thus the automated and easy way is almost certainly the best way to go for Structure-seq2 libraries, as all information is already included. Adapter fragments left on either end of the read may prevent proper read mapping, and adapter fragments left on the 5' end will render proper RT stop detection impossible. Note that we have as of yet not tested any other short read trimmers in the context of a StructureFold experiment, but they should theoretically work, although the settings may require some optimization.

◄2► Map <.fastq> reads to reference transcriptome

This step maps reads to a reference transcriptome to determine the transcript and position on the transcript that best explains each sequenced read. Transcriptomes with an excessive amount of redundancy between transcripts or isoforms may result in a high degree of read multi-mapping; as noted previously, pruning such transcriptomes before mapping to them will greatly simplify both the data processing and downstream analyses. Bowtie 2 [10] is very efficient, and the batch script that drives it maintains detailed logs of mapping statistics, thus it is possible to try mapping against several versions of the same transcriptome until you find a comfortable optimum for your experiment. Custom built transcriptome assemblies are fine.

2.A Run batch mapping script (recommended for everyone)

Create a Bowtie 2 index for your transcriptome (bowtie2-build). Locate your folder that contains all trimmed <.fastq> files. Run the script called fastq_mapper.py, making sure you have built and input the proper path to the Bowtie2 reference you intend to map against. This will automatically map each <.fastq> in the directory using the recommend settings for a StructureFold2 analysis while keeping the same file nomenclature, as it creates <.sam> files suffixed with '_mapped' from <.fastq> files. Additional options are available in the help menu of the script, which can be accessed with the '-h' option.

bowtie2-build my_transcripts.fa my_transcriptome

Place the four .bt2 files for that transcriptome in a convenient place, perhaps a folder called indexes. Thus when running the next script, the path to use that bowtie 2 index would be /path/to/indexes/my_transcriptome

② fastq_mapper.py

A_trimmed.fastq /path/to/indexes/my_transcriptome → A_trimmed_mapped.sam

There are a number of options to consider when running this script. The -threads option will allow the use of more computing power to map faster, while the -log option will provide a detailed <.csv> of read mapping statistics. If you do not wish to store unmapped reads in the <.sam> file, use the -nofails option. By default, secondary alignments of reads (multi-mapped reads) are allowed, but may be disabled at this step with the -nomulti option. Larger plant transcriptomes present the unique challenge of a history of nested polyploidy, while mammalian transcriptomes may contain an excessive amount of very similar transcript isoforms due to the prevalence of alternative splicing; thus, it is up to the user to determine the appropriate amount of multi-mapping tolerated given the transcriptome of interest.

Options
-h, --help   show this help message and exit
-phred64   Use phred64 quality scores instead of phred33
-nomulti   Do not accept multimaps (bowtie option -a off)
-nofails   Do not log failed mappings (bowtie option --no-unal on)
-log   Create an explicit log of the mappings
-threads THREADS   [default = 4] Number of threads to use
-logname LOGNAME   [default = batch_log.csv] name of the log file
-suffix SUFFIX   [default = mapped] SAM file suffix

2.B Run a custom short read mapper

Run any short read aligner you want and collect the <.sam> files for the next step. If your aligner of choice creates <.bam> files instead, use samtools to convert these files to <.sam> files. Note that we have as of yet not tested any other short read aligners in the context of a StructureFold2 experiment, but they should theoretically work, although the settings may require some optimization.

◄3► **Filter <.sam> read mappings**

This step is designed to eliminate undesirable read mappings. StructureFold2's default paradigm discards reads mapped to the reverse strand of the transcript, reads that have a mismatch on the first 5' mapping base, and reads with more than 3 total mismatches/indels from the reference sequence. Depending on your chosen reference transcriptome, the length and quality of your reads,

or any other unforeseen biology, these settings are configurable to accommodate your experiment.

3.A Run batch filtering script (recommended for everyone)

Run sam_filter.py in the directory where all of the <.sam> files generated from mapping are located. This will run a SAMtools command as well as manually edit the files to filter out reads that are undesirably mapped. New filtered files are appended with an additional suffix ('filtered' by default). Additional options are available in the help menu of the script, which can be accessed with the '-h' option.

③🐍 sam_filter.py

A_trimmed_mapped.sam → A_trimmed_mapped_filtered.sam

We highly recommend using the default settings for this step. If you do not wish to have a detailed log of all the filtering at this step, the -turbo flag will process the reads much faster. If you do not wish to work on every <.sam> in the directory, use the -sam flag to indicate specific files to operate on. Although the suffix may be changed with the -suffix flag, we recommend keeping the set pattern of suffixes, i.e. trimmed, mapped, filtered, such that the status and progress of every sample is readily discernible by name and extension. The -max_mismatch setting is configurable; longer reads or transcriptome quality may require relaxing this setting from its default of 3.

Options
-h, --help	show this help message and exit
-turbo	All filter options ignored, default settings, no log
-sam SAM [SAM …]	Specific files to operate on
-keep_all	Keep all intermediate files
-keep_reverse	Keep mappings from the reverse strand
-remove_secondary	Remove secondary alignments
-allow_bp1_mismatch	Accept mappings with first base mismatches
-logname LOGNAME	[default = filter_log.csv] Name of the log file
-suffix SUFFIX	[default = filtered] filtered <.sam> file suffix
-max_mismatch MAX_MM	[default = 3] Maximum allowed mismatches/indels

3.B The Hard way

Filter your <.sam> files any way you choose. Samtools has very useful commands, and at the minimum you must remove un-mapped reads if there are any left over after mapping, as these cannot be processed by the next steps. Unless <.sam> are processed by the filter, there is no guarantee they will work in the next steps. However <.sam> format is universal, so regardless of the read mapper used, the output should be usable. At this time StructureFold2 does not support the use of paired-end reads, and cannot filter <.sam> reads with bitflags from paired-end reads.

◄4► Generate <.rtsc> from filtered read mappings <.sam>

This step will use each individual <.sam> file to generate a corresponding <.rtsc> file, which details the number of times where reverse transcriptase stopped at each base along each transcript. Multiple <.rtsc> that are replicates of the same biological sample are then combined: the individual replicates may be compared to assess repeatability.

4.1 Generate <.rtsc> from filtered read mappings <.sam>

Run sam_to_rtsc.py. The easiest way to run the script is to simply move all of the unfiltered <.sam> files to another directory, and run the script in a directory only retaining the filtered <.sam> files. The second way is to provide a suffix, such that this process only operates on <.sam> files with that suffix, e.g. 'filtered'. Alternatively, the script may be run on an individual file. Finally, the -trim option allows removal of the accumulated suffix chain, such that further manipulating the <.rtsc> files will be more intuitive.

④ sam_to_rtsc.py

A_trimmed_mapped_filtered.sam -trim _trimmed_mapped_filtered index.fasta→ A.rtsc

This step requires the <.fasta> file used to generate the Bowtie 2 index which the reads were mapped against. The new <.rtsc> files should be very compact compared to the <.sam> files, thus more portable and easier to analyze. While not final reactivity scores, the <.rtsc> files may be analyzed independently to observe natural modifications and stops, e.g. in the minus structure-probing chemical control libraries.

Options
-h, --help   show this help message and exit
-single SINGLE   Operate on this single file, rather than the directory
-suffix SUFFIX   Operate only on <.sam> with this suffix before the extension
-trim TRIM   Remove this suffix from output file name before writing

4.2 Combine <.rtsc> replicates of a single treatment into a single <.rtsc> file

Run rtsc_combine.py, selecting the files/replicates you wish to combine, and placing them into the script as arguments. Although this effectively combines the <.rtsc> files, keeping the individual replicates will allow you to do a repeatability analysis in subsequent steps, plus the <.rtsc> files are small enough that it doesn't really cause problems to keep all of them around.

① rtsc_combine.py

A1.rtsc, A2.rtsc, A3.rtsc → A.rtsc

If you wish to begin descriptively naming your files at this point, the merged <.rtsc> files are a good starting point; use the -name argument to specify the output filename, where the <.rtsc> extension will be added automatically if it is not included.

Options
-h, --help   show this help message and exit
-sort   Sort output by transcript name
-name NAME   Specify output file name

4.3 Calculate transcript coverage

Run rtsc_coverage.py either in batch mode (default) or on single <.rtsc> files; it will calculate the number of stops that occur on every base given for a nucleotide specificity (default AC for DMS) divided by the number of those bases in the transcript. This requires supplying the script with the <.fasta> file used for making the Bowtie 2 index.

Ⓙ 🐍 rtsc_coverage.py

A.rtsc → A_coverage.csv

The general case only requires generating coverage files for your reagent-treated libraries, as it is the overlap of these reagent-treated library coverages that ultimately determines which transcripts are 'resolvable' in downstream analyses (Figure 3). Thus generating single files for the reagent treated libraries to enable calculating this (+) reagent overlap in the next step will generate a useful list for filtering, but a more authoritative <.csv> file of coverage information for all samples may be useful for other things. Thus it is recommended that you generate one file for all (+) reagent coverages.

Options
-h, --help   show this help message and exit
-single SINGLE   Operate on this single file, rather than the directory
-bases BASES   [default = AC] Coverage Specificity

4.4 Generate coverage overlap list

This step will take one or more coverage files containing one or more coverage profiles and generate a flat list file containing only the transcripts with greater than n coverage among all represented samples. This list may be generated at a user-defined threshold (-n option) and is used by downstream scripts to filter out insufficiently covered transcripts.

Ⓝ 🐍 coverage_overlap.py

A_coverage.csv, D_coverage.csv → A_D_overlap_1.txt

The script can take all coverage <.csv> files in the directory (those that end with coverage.csv), or take any number of individual <.csv> files with the -f option. If two files share any of the same named columns between them, the first instance of that column (alphabetical in directory mode, input order if using specific <.csv> files) will be overridden by the second, i.e. if you batch-generated a coverage <.csv> file for the entire directory, and also generated single files with the coverage of each individual sample, there would be a replicated column of each single file's coverage in the larger file.

The typical case is going to be taking the coverage of all (+)DMS (or other structure-probing reagent) libraries from both a control (no biological treatment) and any biological treatment sample(s), and testing for their coverage overlap over a given threshold (Figure 3), i.e. generate a list containing the transcripts with over 1 coverage in all reagent-treated libraries in all conditions which are to be compared. Thus the simplest method would be to generate coverage files, only for (+) reagent, one library at a time in step 4.4 and test for overlap only of those files in this step for every contrast to be included. If you wish to incorporate (-) chemical reagent coverage into your criteria, simply test for those overlaps.

Options
-h, --help   show this help message and exit
-n THRESHOLD   [default = 1.0] coverage threshold to use
-f CSVS [CSVS …]   Specific <.csv>s to use
-batchdir   Use all coverage <.csv> in the directory

4.5 Analyze replicate correlation (Optional)

For each group of replicate <.rtsc> files, generate/reformat the data between replicates using rtsc_correlation.py, which generates an easy to use <.csv> file. You can also calculate the correlation between any other <.rtsc>, or combined <.rtsc> files.

Ⓚ🐍 rtsc_correlation.py

A.rtsc, B.rtsc, C.rtsc → A_B_C_correlation.csv

The <.csv> file is readily readable by statistical software, although we highly recommend the use of R. R scripts may be included in future versions of StructureFold2 for common analyses. You may decide to run a transcriptome-wide correlation i.e. compare the number of stops on every base in the transcriptome between two replicates, or you may decide to subdivide the correlation into transcripts; thus indicating how correlated the RT stops per transcript are between replicates.

```
Options
-h, --help              show this help message and exit
-sort                   Sort output by transcript name
-name NAME              Specify output file name
-fasta FASTA            <.fasta> to apply specificity
-spec SPEC              [ACGT] Nucleotide Specifictiy
-restrict RESTRICT      Filter to these transcripts via coverage file
```

4.6 Analyze RT stop specificity (Optional)

In order to check that both the reagent minus and reagent plus samples are displaying distinct modification patterns, the RT stop specificity of either individual or combined <.rtsc> may be queried separately by file or together by directory. This will require the reference <.fasta> as a reference.

Ⓛ🐍 rtsc_specificity.py

A.rtsc, B.rtsc, C.rtsc → A_B_C_specificity.csv

The resultant <.csv> has both raw counts and fractional percentages for the RT stops of each nucleotide.

```
Options
-h, --help              show this help message and exit
-index INDEX            <.fasta> file used to generate the <.rtsc>
-rtsc RTSC [RTSC ...]   Operate on specific <.rtsc>
-name NAME              Specify output file name
```

4.7 Generate transcript abundance (Optional)

Although not as authoritative as a separate RNA-seq experiment, the relative abundances of transcripts can be approximated by summing RT stop counts per transcript, yielding RTSC hits per kilobase per million reads (RTPKM).

Ⓜ🐍 rtsc_abundances.py

The resultant <.csv> has a RTPKM value for every transcript in the <.rtsc>.

Options
-h, --help   show this help message and exit
-rtsc RTSC [RTSC …]   Operate on these files, rather than the directory

◄5► Generate <.react> from RT stop counts <.rtsc>

In this step, we will take a (-)reagent and a (+)reagent combined <.rtsc> file and the reference transcriptome to derive per base reactivity, i.e. generate a <.react> file. This script will also output a <.scale> file which is necessary as a common normalization scale to be used in generating all subsequent <.react> files that will be compared to the first generated file.

5.1 Generate <.react> from <.rtsc>s

Execute the script with specific -/+ reagent files and the reference transcriptome <.fasta>. This will yield a <.react> file and a <.scale> file. While the defaults are recommended, we have included several additional options for this step. When the intention is to compare directly measured chemical reactivity patterns between samples rather than to compare processed reactivity patterns or folded RNAs generated with reactivities as restraints , applying the 2-8% normalization scale or use of the natural log may overprocess the data. In such cases, disabling these options may offer a more precise look at the raw modification signal; The option -ln_off will remove the ln from equations (1) and (2) and without adding 1 to the raw RT counts and -nrm_off likewise disables the 2-8% normalization. The -threshold option changes the reactivity cap (default 7). For alternative structure-probing reagents, the nucleotide specificity of the reagent may be defined with -bases; the default is for DMS (AC). Since the default settings use both ln and 2-8% normalization, they are suffixed to the output file, but will not be if these options are turned off.

⑤ rtsc_to_react.py

A.rtsc, B.rtsc seq.fasta→ A_B_ln_nrm.react, A_B_ln_nrm.scale

$$P(i) = \frac{\ln[P_r(i)+1]}{\{\sum_{i=0}^{l} \ln[P_r(i)+1]\}/l} \tag{1}$$

$$M(i) = \frac{\ln[M_r(i)+1]}{\{\sum_{i=0}^{l} \ln[M_r(i)+1]\}/l} \tag{2}$$

$$\theta(i) = max[P(i) - M(i), 0] \tag{3}$$

Here, $P_r(i)$ and $M_r(i)$ are the raw 'r' numbers of RT stops mapped to nucleotide i (all four nucleotides are included) on the transcript in the plus (P) and minus (M) chemical probing reagent libraries, respectively, and l is the length of the transcript. $P_r(0)$ and $M_r(0)$ are the raw numbers of 5′-runoff RT reads. $\theta(i)$ is the raw DMS reactivity for nucleotide i.

Transcripts that are unable to produce an internal normalization scale due to an insufficient number of RT stops are not reported in the output <.react> file. A log of these transcripts may be generated by invoking the -save_fails option. Output may be further restricted by

providing a flat list of transcripts to include via the -restrict option.

Options
-h, --help   show this help message and exit
-threshold THRESHOLD   [default = 7.0] Reactivity Cap
-ln_off   Do not take the natural log of the stop counts
-nrm_off   Turn off 2-8 % normalization of the derived reactivity
-save_fails   Log transcripts with zero or missing scales
-scale SCALE   Provide a normalizaiton <.scale> for calculation
-bases BASES   [default = AC] Reaction Specificity, (AGCT) for SHAPE
-name NAME   Change the name of the outfile, overrides default
-restrict RESTRICT   Limit analysis to these specific transcripts <.txt>

5.2 Generate normalized <.reacts> from <.rtsc>s

For each condition which is to be compared to the 'control' <.react> file, run the script again using the option to provide your own normalization scale (-scale), inputting the <.scale> file generated under the control conditions. If a <.scale> file is provided but 2-8% normalization is turned off (-nrm_off) transcripts missing from the scale file will be excluded from the output.

⑤ 🐍 rtsc_to_react.py

C.rtsc, D.rtsc seq.fasta -scale A_B_ln_nrm.scale → C_D_ln_nrm.react

Once you have generated your <.react> files, the core StructureFold2 pipeline is finished, largely leaving the user to determine the exact nature of her or his analysis. Many of the auxiliary scripts will accelerate or augment common follow up analyses and are detailed later in the manual. Common or preliminary analyses include using the <.react>s as restraints to batch fold RNAs using the RNAStructure package [5] (or Vienna package [6]) via the batch_fold_RNA.py script, or looking at reactivity changes directly via the reactivity_statistics.py script, which creates an easy to use <.csv> file of many metrics between conditions. Any two <.react> files that are going to have any of their downstream data compared should share the same <.scale> if one is normalizing.

◄Auxiliary Steps►

This part of the analysis is more up to the user. There are two general ways of analyzing data from here on out that are complementary yet different (Figure 2). The top or Green Path first folds the RNA using the restraints generated by your chemical probing data via the RNAStructure package [5], then extracts metrics from folded structures. The bottom or Blue Path instead works directly on the transcript reactivity patterns, extracting metrics such as average reactivity, Gini of reactivity, or RMSD (Root Mean Square Deviation) of reactivity to probe more directly for change of reactivity between conditions. While some tools are shared between each analysis path, the end result of each path is <.csv> files, which may be easily combined and analyzed in R or any other mathematics/statistics suite, although they can also be used separately. Thus, both folded and raw metrics can be easily combined for an integrated analysis. Only reactivity_windows.py outputs something that is not a <.csv>, namely a <.fasta> file, which may be used in MEME [11] or other similar types of analyses.

Reactivity files (<.react>) may be subdivided into separate transcript features <.react>s before continuing, e.g. 5'UTR, CDS, and 3'UTR. Due to the variance in both quality and formatting of transcript annotation, we do not have a standardized way to perform such subdividing at this time. We recommend that you create separate <.react> files for each gene feature you wish to

analyze independently by parsing the whole transcript <.react>, while also keeping the 'whole transcript' files. It is inadvisable to map exclusively to features and generate files that way, as truncated CDS and UTR regions may not actually allow reads that span the feature border to accurately map.

<u>A - Green Path</u>

Folding is a computationally intensive procedure, thus one should anticipate dedicating a machine for at least ~12 hours to batch fold ~6000 transcripts, and transcripts will need to be folded once for every condition your experiment contains. Shorter transcripts will fold much faster.

<u>A.1 – Fold RNAs, predict structures</u>

This step will require a <.react> file to be used as folding restraints, as well as the corresponding <.fasta> file and the list of transcripts you wish to fold – this last file is most often generated in step 4.4, but could be any arbitrary flat list of transcripts that are present in both the <.react> and the <.fasta> files.

Ⓐ🐍 batch_fold_RNA.py

A_coverage_1.txt transcripts.fasta 1 -r A.react → [predicted structures]

This will yield a new directory with two sub-directories; one full of <.ct> or connectivity tables for each transcript, and one full of <.ps> or postscripts for each transcript. CT files contain a matrix detailing all the predicted base pair interactions in the RNA secondary structure. Postscript files contain a pictorial representation of an RNA secondary structure. By default, every transcript is folded into exclusively the MFE or minimum free energy structure, for which all of the Green Path scripts are designed to work with. This saves on computational time and prevents overly complicating the genome-wide aspect of the analysis. To report all predicted structures, use the –multiple option, but the variable number of potential structures for each transcript will make it impossible to use the subsequent scripts, which all assume one metric or comparative metric per transcript. However, exploration of all potential structures is practical for a more directed analysis on a limited number of candidate structures.

Options
-h, --help   show this help message and exit

-T TEMPERATURE   --Temperature TEMPERATURE
                                 The temperature under which the RNA structures are predicted
                                 [Default 310.15 K]

-r CONSTRAINT_FILE   --reactivity CONSTRAINT_FILE
                                 Reactivity file (.react file) used as constrants from prediction

-th THRES   --threshold THRES
                                 Threshold for reactivity. Any nucleotide with reactivity over
                                 threshold will be set as single stranded. Once threshold is set,
                                 reactivities will be converted into hard contraints

-sm SLOPE   --slope SLOPE
                                 Slope for RNA structure prediction with restrains [Default 1.8]
                                 (Parameter for RNAstructure, only for prediction using

RNAstructure)

-si INTERCEPT,	--intercept INTERCEPT Slope for RNA structure prediction with restrains [Default -0.6] (Parameter for RNAstructure, only for prediction using RNAstructure)
-sht N, --shift N	Ignore the reactivities on the last N nucleotide of each RNA to be predicted [Default 0]
-minl MINL,	--minumum_length MINL The minumum length of the RNA required for folding [Default 10]
-maxl MAXL,	--maximum_length MAXL The maximum length of the RNA required for folding [Default 5000]
-par, --PAR	Calculate partition function and output base-pairing probabilities instead of RNA structures
-mul, --multiple	Output multiple predicted RNA structures instead of just outputing the MFE structure (only for RNAstructure prediction)
-p P, --process P	Number of threads for parallel computing [Default 1]
-md MD	--maxdistance MD Specify a maximum pairing distance between nucleotides [Default: no restraint]

A.2 – Calculate structure strandedness

Running another script on the directory of <.ct> files will generate a convenient <.csv> file containing the percentage of bases that are double stranded in each predicted structure. This information may be analyzed independently or combined with other <.csv> data for an integrated analysis.

Ⓑ make_strand_csv.py

ct_directory → stranded.csv

Options
-h, --help	show this help message and exit
-name NAME	Specify output file name
-suffix SUFFIX	Append given suffix to data column names

A.3 – Calculate PPV between two conditions

PPV is defined as the positive predictive value. While nominally this is designed as a metric to compare how close a predicted structure is to a known structure, it will assess the percentage of base pairs in the predicted (treatment) structure that are also present in the known (control) structure. Running the PPV script will take two directories of <.ct> files and will compare the structure of every transcript that occurs in both conditions, generating a comparative PPV value, and logging this information to a convenient <.csv> file.

ⓒ🐍 make_PPV_csv.py

ct_directory_1 ct_directory_2 → ppv.csv

Options
-h, --help   show this help message and exit
-n NAME   [default = stats.csv] outfile
-suffix_1 SUFFIX_1   Suffix from directory 1, if files do not share name suffix
-suffix_2 SUFFIX_2   Suffix from directory 2, if files do not share name suffix

A.4 – Gather MFEs of predicted structures

Running another script on the directory of <.ct> files will generate a convenient <.csv> file of all the structures' free energies. This information may be analyzed independently or combined with other <.csv> data for an integrated analysis.

Ⓓ🐍 make_MFE_csv.py

ct_directory → mfe.csv

Options
-h, --help   show this help message and exit
-name NAME   [default = MFE.csv] Output file name

B - Blue Path

Calculating and combining all of the derived statistics from <.react> files is a rather straightforward and not computationally demanding process. Each of these steps will generate a <.csv> file, which may be combined together into one <.csv> file for an integrated analysis, or analyzed entirely by itself. Investigating patterns and changes in the reactivity data presents a complementary avenue to working with predicted structures.

B.1 – Calculate basic transcript reactivity statistics

To gather simple statistics, simply execute the reactivity_statistics.py script in a directory with one or more <.react> files. You may provide a coverage overlap file (step 4.4, Figure 3) to filter out transcripts for which statistics should not be generated due to inadequate coverage. We recommend using a coverage overlap file generated with coverage at or above 1 within all (+)reagent samples that were integrated into the <.react> files being assayed. For more information on coverage overlap files, see Figure 3. This will yield a <.csv> file of appropriately named columns for average reactivity, standard deviation of reactivity, Gini index of reactivity, and max reactivity, for every such transcript. Additional options are included to ignore transcripts that are too short (default 20) and ignore the extreme 3' end of transcripts (default 30 nt); both of these options are configurable and logged in the output file's name suffixes. The extreme 3' ends of transcripts are often depleted of RT stops because the 5' end of the read defines the stop, these regions tend only to lower average reactivity values while increasing sd and Gini across the board, hence lowering precision. The coverage <.csv> file of any/all samples may be combined with this <.csv> file for extra filtering options.

Ⓔ🐍 react_statistics.py

control.react condition.react → statistics.csv

Another method to do this is simply to merge a <.csv> of coverages from every sample to a stats file generated without any coverage overlap restrictions and do all coverage filtering when analyzing the downstream <.csv>, rather than when generating it. In these cases we still recommend using (+) probing reagent coverage as the threshold metric. More advanced users may find this method more to their liking as data are never really discarded, and iterations of thresholds can explored more readily. In this case, using the batch functionality in step 4.3 will yield a concise <.csv> file of coverages in all samples.

Options
-h, --help   show this help message and exit
-react REACT [REACT ...]   Operate on specific <.react> files
-restrict RESTRICT   Filter to these transcripts via coverage file
-name NAME   Specify output file name
-n TRIM   [default = 20] ignore n last bp of reactivity
-m MINLEN   [default = 10] minimum length of transcript

B.2 – Calculate comparative transcript RMSD

RMSD or Root-Mean-Square Deviation is a measure that can be applied to any two vectors of reactivity of the same length, i.e. the same transcript in two conditions, measuring the amount of shift between the two vectors, or in this case, the rearrangement of reactivity between conditions. This is implicitly a pairwise operation, thus two <.react> files and a coverage overlap file (optional but highly recommended, to only perform the comparison between transcripts adequately covered in both conditions) are used in this step to create a <.csv> of RMSD values. Normalized RMSD (NRMSD), normalized by the combined average reactivity, may also be calculated by invoking a script option.

Ⓕ 🐍 reactivity_RMSD.py

control.react condition.react → RMSD.csv

Options
-h, --help   show this help message and exit
-normalize   Normalize output RMSD values
-name NAME   Specify output file name
-restrict RESTRICT   Restrict output to these transcripts

B.3 – Compare transcript reactivity maxima

Another way to compare transcript reactivity between conditions is to see how the most reactive points have shifted, i.e. how many of the reactivity maxima are shared between conditions. This is an implicitly pairwise operation, thus two <.react> files and a coverage overlap file (optional but highly recommended, to only perform the comparison between transcripts adequately covered in both conditions) are used in this step to create a <.csv> of the number of shared maxima. The script may be configured to look for more or fewer maxima, or include a bit of 'wiggle' in allowing maxima to be off by n nucleotides. If there are fewer than the input number of maxima in either <.react> file of a particular transcript, NA will be logged for the transcript's value of shared maxima.

Ⓖ 🐍 react_maxima.py

control.react condition.react → maxima.csv

Options
-h, --help	show this help message and exit
-restrict RESTRICT	Limit analysis to these specific transcripts
-maxima MAXIMA	[default = 20] Number of maxima to pick from both transcripts
-wiggle WIGGLE	[default = 3] Number of bases maxima can be off by

B.4 – Compare transcript reactivity windows

To investigate and find particular short motifs of transcripts that change reactivity between conditions, use the react_windows script. This is an implicitly pairwise operation, thus two <.react> files and a coverage overlap file (optional but highly recommended, -restrict, to only perform the comparison between transcripts adequately covered in both conditions) are used in this step. First, the script will walk across all transcripts with windows of n length (setting -wlen), by steps of x length (setting -wstep), gathering the net change and total change (absolute value of change) across each of these steps. By default it will write out all of these windows to a file; however, you may filter these results, taking the top y% (option -perc) of either the reactivity losses, gains, or total change (options -filter_loss, -filter_gain, -filter_delta), and the script will automatically suffix the output according to the filter(s) used. Additionally, the sequences may be output in a separate <.fasta> file (option -fastaout) which could be used in a few ways, such as for MEME analysis.

Ⓗ react_windows.py

control.react condition.react sequences.fasta → windows.csv, windows.fasta

Options
-h, --help	show this help message and exit
-wlen WLEN	[default = 50] Window Length
-wstep WSTEP	[default = 20] Window Step
-outname OUTNAME	Change the name of the outfile, overrides default
-restrict RESTRICT	<.txt > Limit analysis to these specific transcripts
-filter_loss	Restrict output to largest reactivity losses
-filter_gain	Restrict output to largest reactivity gains
-filter_delta	Restrict output to most changed reactivity
-perc PERC	[default = 25] Filter to this percent of windows
-fastaout	Write windows to <.fasta> format as well

B.5 – Reactivity binning

Binning may find common trends of reactivity distributions along the length of one or more transcripts, or may be useful to compare reactivity distributions along transcripts between conditions to find novel differences or rearrangements. The script to help bin these values offers several options and strategies of binning, reducing long transcripts down into a number of averaged bins of a given length.

Ⓟ react_bins.py

sample.react → single_sequence.csv OR multiple_sequences.csv OR all_sequences.csv

Changing various settings will allow the user to highly customize this analysis. First the minimum required bin size (-size) and number of bins (-bins) the transcript(s) are broken into can be changed. A single transcript may be pulled from the <.react> file (-single), or a specified list of transcripts may be collectively binned together (-multi); in this case the average of each numbered bin is thus reported, and any transcripts in a list that fail to meet these criteria are not included (i.e. too few values in one or more bin). The -all_transcripts command will try to combine all transcripts in the <.react> file and collectively bin them. Thus, either by providing a specific list or by pre-filtering the <.react> file, a class of genes may be separated and binned to probe for reactivity patterns along their length.

Options

-h, --help	show this help message and exit
-bins BINS	[default = 100] bins to create
-size SIZE	[default = 10] minimum number of values per bin
-single SINGLE	Target transcript to bin from the file
-multi MULTI	Flat <.txt> of transcripts to bin, one per line
-all_transcripts	Amalgamate all transcripts in the <.react> file

B.6 – Reactivity delta by position

Many experimenters will have questions centering around the 5' or 3' ends of transcripts, as there are many interesting regulatory RNA structures in these mostly non-coding segments. The react_delta.py script will allow the user to examine these changes on one, several, or all transcripts between two <.react> files, probing a set number of bases from either the 5' end or the 3' end of the transcript for patterns of changes in reactivity. For the transcript(s) analyzed, five values are reported for each base in the segment in the resultant <.csv> file: the relative position of the base (positive indexes for 5', negative indexes for 3'), the average delta at that base (sum of all changes divided by total number of observations at that base including zeros), the average change at that base (sum of all changes divided by number of observations at that base excluding zeros), the average positive change at that base (sum of all positive changes at that base divided by the number of positive changes at that base) and finally the average negative change (sum of all negative changes at that base divided by the number of negative changes at that base).

react_delta.py

control.react, experimental.react → delta.csv

There are several options available when using this script. To indicate which transcripts to use, -single for one specific transcript, -multi with a list of transcripts in <.txt> format, one transcript per line, or -all_transcripts to do this for all transcripts shared between the two <.react> files. By default, it will take 100 bases (configurable with -n) from the 5' end (-tp to use the 3' end). The output <.csv> will have as many rows as base pairs being investigated, plus a header.

Options

-h, --help	show this help message and exit
-n N	Number of bases, default 100
-tp	Start from the 3' end (5' default)
-raw	Do not average changes
-single SINGLE	Target transcript to bin from the file
-multi MULTI	Flat <.txt> of transcripts to bin, one per line
-all_transcripts	Use all transcripts in the <.react> file

B.7 – Transcript composition

Another aspect that can be integrated into an analysis is the nucleotide composition of each transcript or transcript segment being analyzed. To quickly get a convenient <.csv> of the nucleotide composition, simply run the script on your <.fasta> file of interest.

Ⓡ🐍 fasta_composition.py

sequences.fasta → sequences_composition.csv

This will yield a <.csv> with the following fields: transcript, GC_content, AT_content, AC_content, A_content, C_content, G_content, T_content, transcript_length.

Options
-h, --help   show this help message and exit
-single SINGLE   Operate on this single file
-suffix SUFFIX   [default = composition] <.csv> file suffix

◄**Concluding Remarks**►

We aim to offer support and updates on a semi-regular basis as more analyses are requested or developed. Questions, comments, bug-fix requests, or requests for more features may be directed to David Tack (kujiratan@gmail.com). The entire staff of StructureFold2 understands that every experiment is unique; while each tool has been designed to be as modular and accommodating as possible, we will do our best to ensure they will work for everyone.

◄References►

[1] Python Software Foundation. Python Language Reference, version 2.7. http://www.python.org.

[2] Y. Ding, C.K. Kwok, Y. Tang, P.C. Bevilacqua, S.M. Assmann, Genome-wide profiling of in vivo RNA structure at single-nucleotide resolution using structure-seq, Nat Protoc 10(7) (2015) 1050-66.

[3] Y. Ding, Y. Tang, C.K. Kwok, Y. Zhang, P.C. Bevilacqua, S.M. Assmann, In vivo genome-wide profiling of RNA secondary structure reveals novel regulatory features, Nature 505(7485) (2014) 696-700.

[4] L.E. Ritchey, Z. Su, Y. Tang, D.C. Tack, S.M. Assmann, P.C. Bevilacqua, Structure-seq2: sensitive and accurate genome-wide profiling of RNA structure in vivo, Nucleic Acids Res 45(14) (2017) e135.

[5] J.S. Reuter, D.H. Mathews, RNAstructure: software for RNA secondary structure prediction and analysis, BMC Bioinformatics 11 (2010) 129.

[6] I.L. Hofacker, W. Fontana, P.F. Stadler, L.S. Bonhoeffer, M. Tacker, P. Schuster, Fast folding and comparison of RNA secondary structures, Monatshefte für Chemie / Chemical Monthly 125(2) (1994) 167-188.

[7] R Development Core Team. R: A language and enviornment for statistical compution, 2008. http://www.R-project.org.

[8] B.L. Aken, P. Achuthan, W. Akanni, M.R. Amode, F. Bernsdorff, J. Bhai, K. Billis, D. Carvalho-Silva, C. Cummins, P. Clapham, L. Gil, C.G. Giron, L. Gordon, T. Hourlier, S.E. Hunt, S.H. Janacek, T. Juettemann, S. Keenan, M.R. Laird, I. Lavidas, T. Maurel, W. McLaren, B. Moore, D.N. Murphy, R. Nag, V. Newman, M. Nuhn, C.K. Ong, A. Parker, M. Patricio, H.S. Riat, D. Sheppard, H. Sparrow, K. Taylor, A. Thormann, A. Vullo, B. Walts, S.P. Wilder, A. Zadissa, M. Kostadima, F.J. Martin, M. Muffato, E. Perry, M. Ruffier, D.M. Staines, S.J. Trevanion, F. Cunningham, A. Yates, D.R. Zerbino, P. Flicek, Ensembl 2017, Nucleic Acids Res 45(D1) (2017) D635-D642.

[9] M. Martin, Cutadapt removes adapter sequences from high-throughput sequencing reads, EMBnet.Journal 17(1) (2011) 10-12.

[10] B. Langmead, S.L. Salzberg, Fast gapped-read alignment with Bowtie 2, Nature methods 9(4) (2012) 357-9.

[11] T.L. Bailey, M. Boden, F.A. Buske, M. Frith, C.E. Grant, L. Clementi, J. Ren, W.W. Li, W.S. Noble, MEME SUITE: tools for motif discovery and searching, Nucleic Acids Res 37(Web Server issue) (2009) W202-8.

StructureFold II

Software to facilitate rapid analysis of genome-wide chemical probing data on RNA structure

(-)Reagent (+)Reagent A, B: Control / C, D: Experimental

A.fastq / B.fastq / C.fastq / D.fastq

Adapter Trimming / Quality Trimming — fastq_trimmer.py (1)

A_trimmed.fastq / B_trimmed.fastq / C_trimmed.fastq / D_trimmed.fastq

Map to Reference Transcriptome — fastq_mapper.py (2)

A_trimmed_mapped.sam / B_trimmed_mapped.sam / C_trimmed_mapped.sam / D_trimmed_mapped.sam

Filter <.sam> — sam_filter.py (3)

A_trimmed_mapped_filtered.sam / B_trimmed_mapped_filtered.sam / C_trimmed_mapped_filtered.sam / D_trimmed_mapped_filtered.sam

Generate <.rtsc> — sam_to_rtsc.py (4)

A.rtsc / B.rtsc / C.rtsc / D.rtsc

Generate <.react> — rtsc_to_react.py (5)

AB.react / CD.react
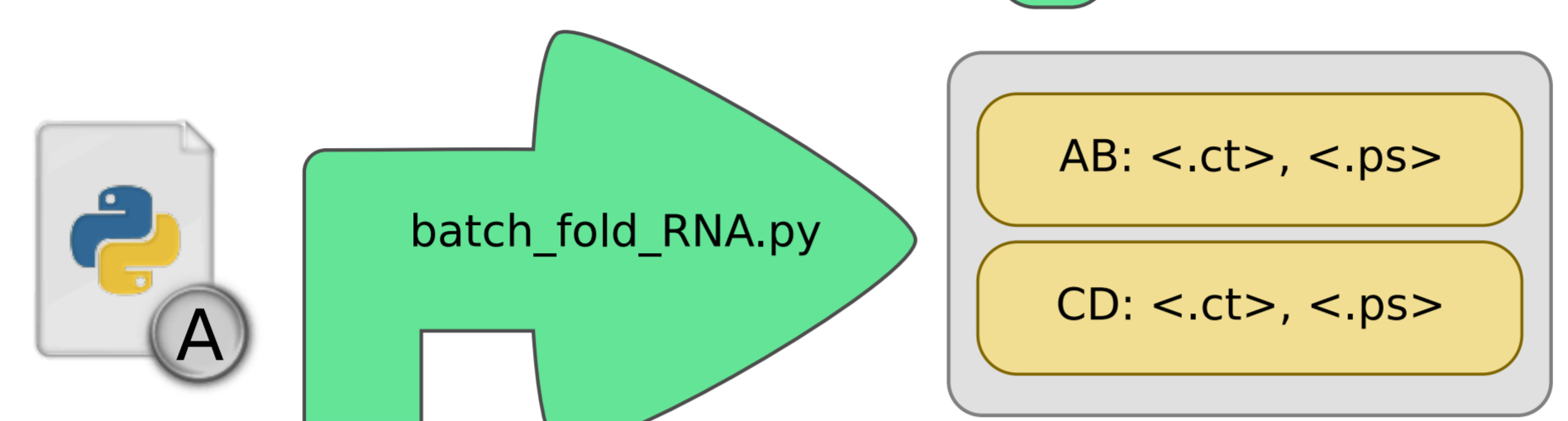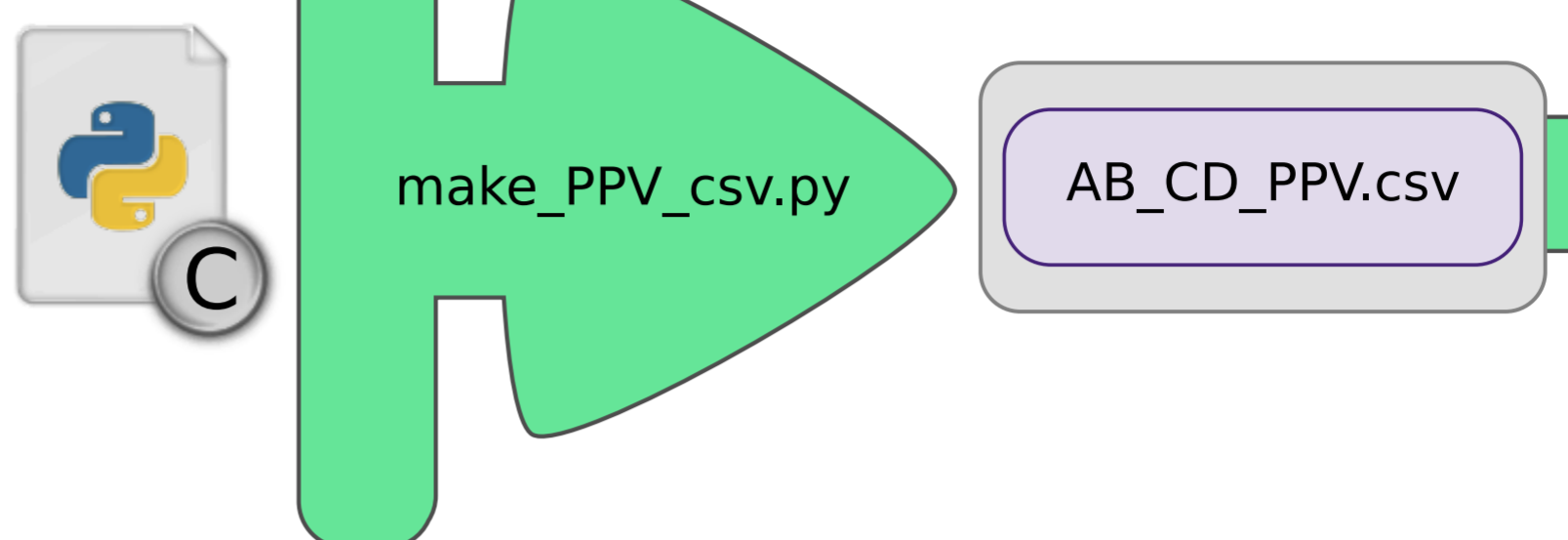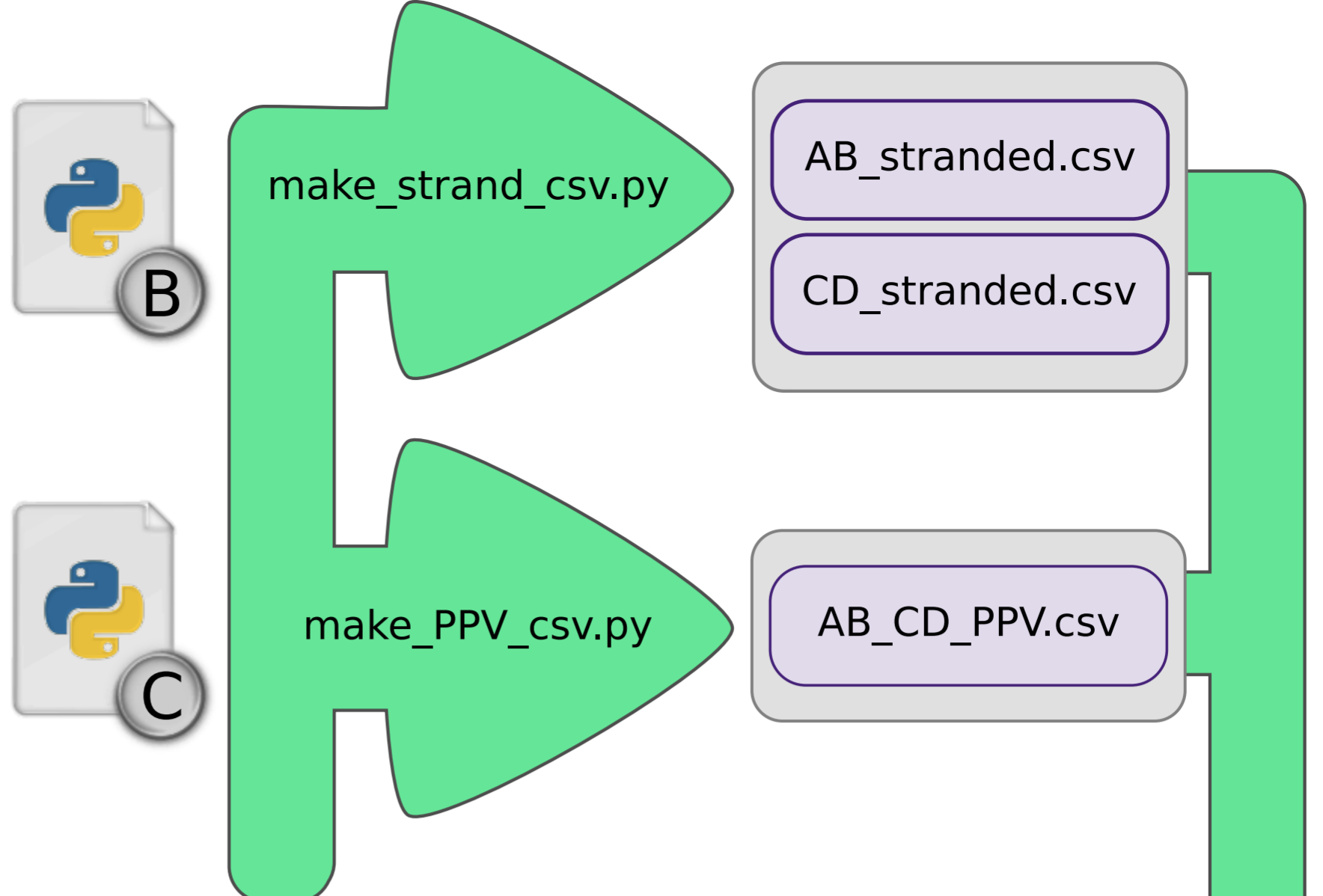
Generating reactivity profiles from sequencing data

Extracting data from <.react> files

<.react> files
Folded RNA files
Data for analysis

make_strand_csv.py
B

AB_stranded.csv
CD_stranded.csv

make_PPV_csv.py
C

AB_CD_PPV.csv

batch_fold_RNA.py
A

AB: <.ct>, <.ps>
CD: <.ct>, <.ps>

make_MFE_csv.py
D

AB_MFE.csv
CD_MFE.csv

AB.react
CD.react
...

react_statisics.py
E

AB_CD_stats.csv

react_RMSD.py
F

AB_CD_RMSD.csv

react_maxima.py
G

AB_CD_maxima.csv

react_windows.py
H

AB_CD_windows.csv
AB_CD_seqs.fasta

Easy to use data!

Filtering transcripts by coverage

B_coverage.csv

D_coverage.csv

F_coverage.csv

coverage_overlap.py

(-)DMS

(+)DMS

A, B: Control
C, D: Experimental 1
E, F: Experimental 2

reactivity_coverage.py

BDF_overlap.txt

reactivity_statisics.py

A.rtsc
C.rtsc
E.rtsc

B.rtsc
D.rtsc
F.rtsc

AB.react
CD.react
EF.react

AB_CD_EF_statistics.csv