# RNA-Hi-C-tools Documentation

***Release 0.3.2***

**Pengfei Yu**

May 15, 2014

CONTENTS

**Contents:**

CHAPTER

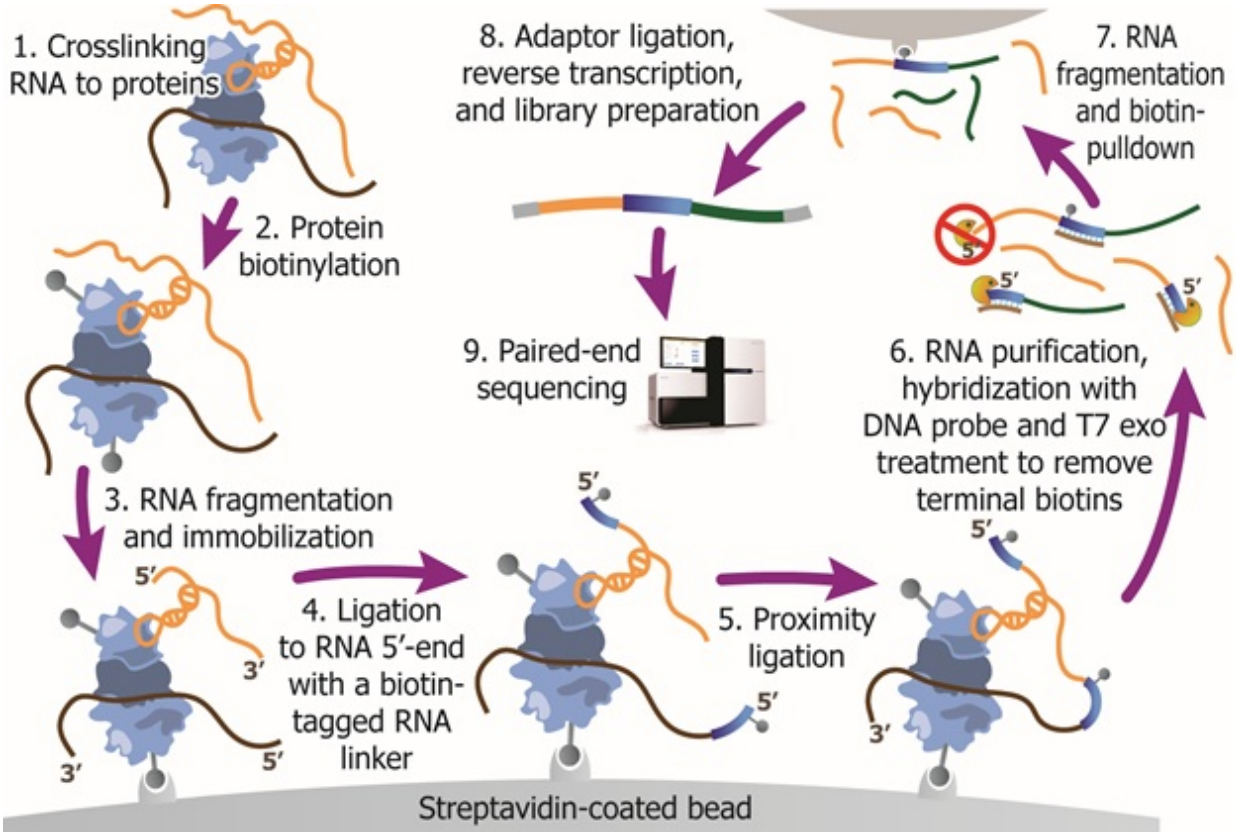
ONE

# RNA-HI-C-TOOLS 0.3 DOCUMENTATION

## 1.1 Overview

**RNA-Hi-C-tools** is a set of bioinformatic tools for analysis of a novel DNA sequencing based technology to detect RNA-RNA interactome and RNA-chromatin interactome (RNA-chromatin interactome is coming soon).

Below is a illustration for the experimental design of this new technology




**See also:**

Offline documentation.

Download a copy of RNA-Hi-C-tools documentation:

- PDF
- Epub

## 1.2 Installation

### 1.2.1 step 1: Install the dependent prerequisites:

1. Python libraries [for python 2.x]:

- Biopython
- Pysam
- BAM2X
- Numpy, Scipy
- Parallel python (Only for `Select_strongInteraction_pp.py`)

2. The Boost.Python C++ library
3. Other softwares needed:

- Bowtie (not Bowtie 2)
- samtools
- NCBI blast+ (use blastn)

### 1.2.2 Step 2: Download the package

Clone the package from GitHub:

```
git clone http://github.com/yu68/RNA-Hi-C.git
```

### 1.2.3 Step 3: Add library source to your python path

Add these lines into your ~/.bash_profile or ~/.profile

```
Location="/path/of/RNA-Hi-C-tools" # change accordingly
export PYTHONPATH="$Location/src:$PYTHONPATH"
export PATH="$PATH:$Location/bin"
Loc_lib="/path/of/boost_1_xx_0/lib/"  # change accordingly
export LD_LIBRARY_PATH="$Loc_lib:$LD_LIBRARY_PATH"
```

## 1.3 Support

For issues related to the use of RNA-Hi-C-tools, or if you want to **report a bug or request a feature**, please contact Pengfei Yu <p3yu at ucsd dot edu>

CHAPTER

TWO

# ANALYSIS PIPELINE

## 2.1 Overview

The next generation DNA sequencing based technology utilize RNA proximity ligation to transfrom RNA-RNA interactions into chimeric DNAs. Through sequencing and mapping these chimeric DNAs, it is able to achieve high-throughput mapping of nearly entire interaction networks. RNA linkers were introduced to mark the junction of the ligation and help to split the chimeric RNAs into two interacting RNAs. This bioinformatic pipeline is trying to obtain the strong interactions from raw fastq sequencing data. The major steps are:

- *Step 1: Remove PCR duplicates.*
- *Step 2: Split library based on barcode.txt.*
- *Step 3: Recover fragments for each library.*
- *Step 4: Split partners and classify different types of fragments.*
- *Step 5: Align both parts of "Paired" fragment to the genome.*
- *Step 6: Determine strong interactions.*
- *Step 7: Visualization of interactions and coverages.*

Other functions:

1. *Determine the RNA types of different parts within fragments.*
2. *Find linker sequences within the library.*
3. *Find intersections between two different interaction sets based on genomic locations*
4. *Find intersections between two different interaction sets based on annotation*
5. *RNA structure prediction by adding digestion site information*

## 2.2 Pipeline

### 2.2.1 Step 1: Remove PCR duplicates.

Starting from the raw pair-end sequencing data, PCR duplicates should be removed as the first step if both the 10nt random indexes and the remaining sequences are exactly the same for two pairs. It is achieved by `remove_dup_PE.py`

```
usage: remove_dup_PE.py [-h] reads1 reads2

Remove duplicated reads which have same sequences for both forward and reverse
reads. Choose the one appears first.
```

```
positional arguments:
  reads1      forward input fastq/fasta file
  reads2      reverse input fastq/fasta file

optional arguments:
  -h, --help  show this help message and exit

Library dependency: Bio, itertools
```

The program will generate two fastq/fasta files after removind PCR duplicates and report how many read pairs has been removed. The output are prefixed with 'Rm_dupPE'

**Note:** One pair is considered as a PCR duplicate only when the sequences of both two ends (including the 10nt random index) are the exactly same as any of other pairs.

### 2.2.2 Step 2: Split library based on barcode.txt.

After removing PCR duplicates, the libraries from different samples are separated based on 4nt barcodes in the middle of random indexes ("RRRBBBBRRR"; R: random, B: barcode). It is implemented by `split_library_pairend.py`

```
usage: split_library_pairend.py [-h] [-f | -q] [-v] [-b BARCODE]
                                [-r RANGE [RANGE ...]] [-t] [-m MAX_SCORE]
                                input1 input2

Example: split_library_pairend.py -q Rm_dupPE_example.F1.fastq
         Rm_dupPE_example.R1.fastq -b barcode.txt

positional arguments:
  input1                input fastq/fasta file 1 for pairend data (contain
                        barcodes)
  input2                input fastq/fasta file 2 for pairend data

optional arguments:
  -h, --help            show this help message and exit
  -f, --fasta           add this option for fasta input file
  -q, --fastq           add this option for fastq input file
  -v, --version         show program's version number and exit
  -b BARCODE, --barcode BARCODE
                        barcode file
  -r RANGE [RANGE ...], --range RANGE [RANGE ...]
                        set range for barcode location within reads,default is
                        full read
  -t, --trim            trim sequence of 10nt index
  -m MAX_SCORE, --max_score MAX_SCORE
                        max(mismatch+indel) allowed for barcode match,
                        otherwise move reads into 'unassigned' file
                        default: 2.

Library dependency: Bio
```

Here is a example for barcode.txt

```
ACCT
CCGG
GGCG
```

The output of this script are several pairs of fastq/fasta files prefixed with the 4nt barcode sequences, together with another pair of fastq/fasta files prefixed with 'unassigned'.

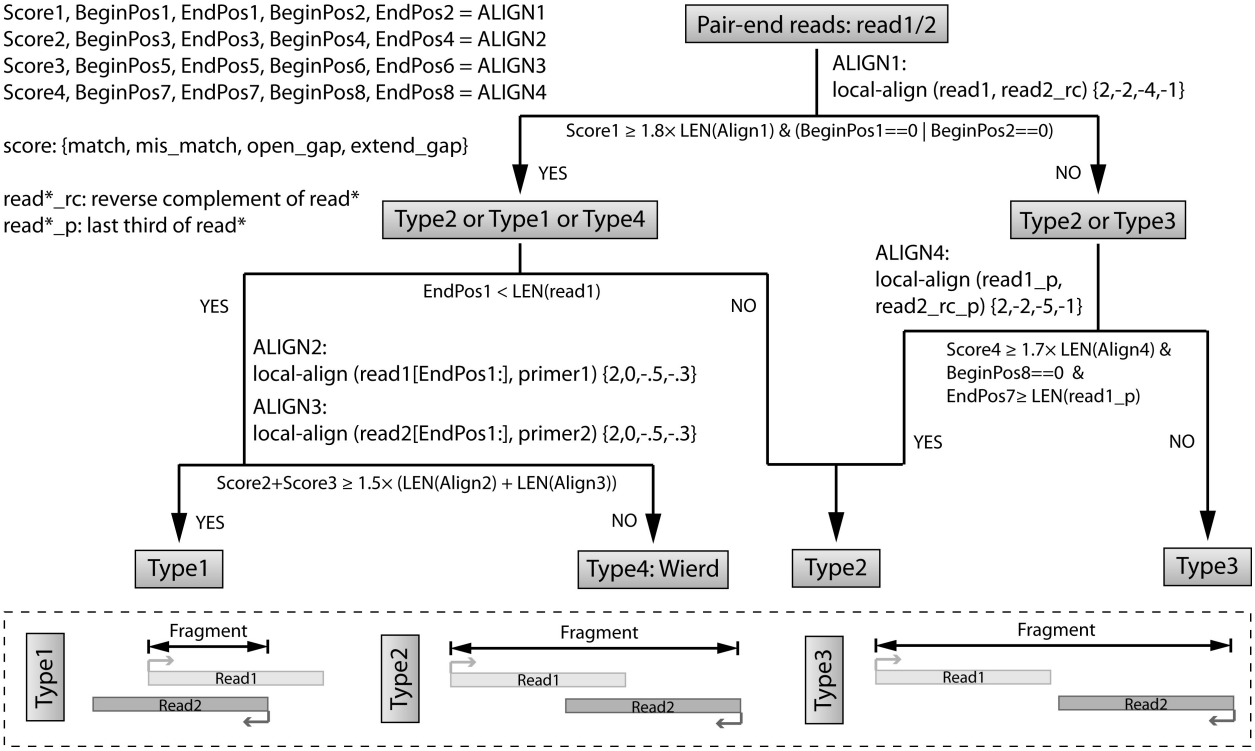
For example, if the input fastq/fasta files are `Rm_dupPE_example.F1.fastq` and `Rm_dupPE_example.R1.fastq`, and the barcode file is the same as above, then the output files are:

- ACCT_Rm_dupPE_example.F1.fastq
- ACCT_Rm_dupPE_example.R1.fastq
- CCGG_Rm_dupPE_example.F1.fastq
- CCGG_Rm_dupPE_example.R1.fastq
- GGCG_Rm_dupPE_example.F1.fastq
- GGCG_Rm_dupPE_example.R1.fastq
- unassigned_Rm_dupPE_example.F1.fastq
- unassigned_Rm_dupPE_example.R1.fastq

### 2.2.3 Step 3: Recover fragments for each library.

**After splitting the libraries, the later steps from here (Step 3-7) need to be executed parallelly for each sample.**

In this step, we are trying to recover the fragments based on local alignment. The fragments are classifed as several different types as shown in the figure below. The flow chart is also clarified at the top.




We will use a complied program `recoverFragment` to do that

```
recoverFragment - recover fragment into 4 different categories from pair-end seq data
================================================================================

SYNOPSIS
```

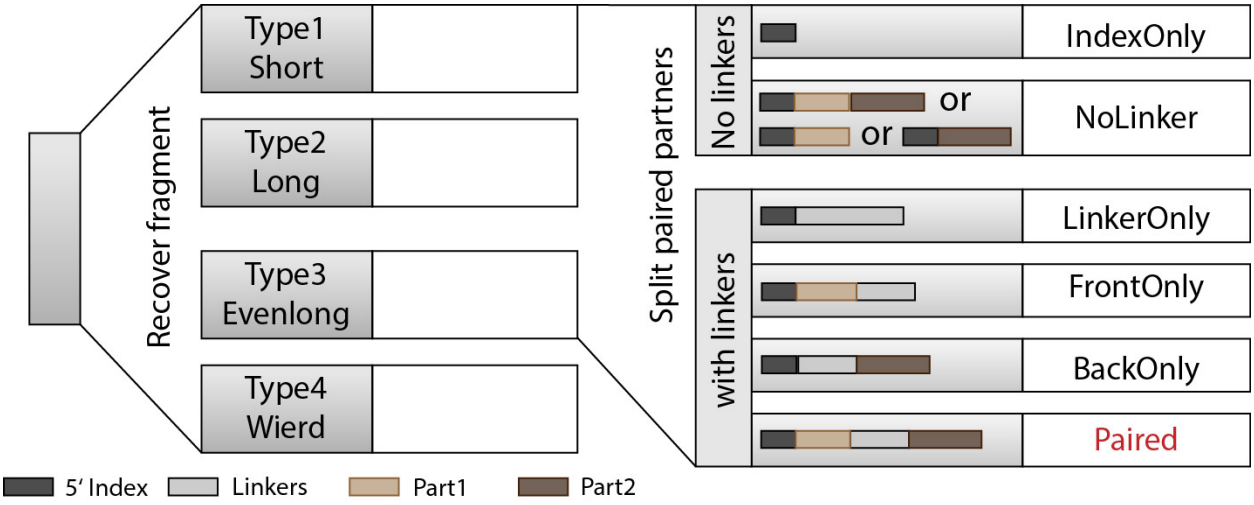
```
DESCRIPTION
    -h, --help
          Displays this help message.
    --version
          Display version information
    -I, --inputs STR
          input of forward and reverse fastq file, path of two files separated by SPACE
    -p, --primer STR
          fasta file contianing two primer sequences
    -v, --verbose
          print alignment information for each alignment

EXAMPLES
    recoverFragment -I read_1.fastq read_2.fastq -p primer.fasta
          store fragment using fasta/fastq into 4 output files
          'short_*', 'long_*','evenlong_*','wierd_*'

VERSION
    recoverFragment version: 0.1
    Last update August 2013
```

### 2.2.4 Step 4: Split partners and classify different types of fragments.

When we recovered the fragments, the next we are goting to do is to find parts that are seprarated by the linkers, and from here, we will be able to classify the fragments into different types: "IndexOnly", "NoLinker", "LinkerOnly", "BackOnly", "FrontOnly", "Paired". (see the figure below).




This will be done by `split_partner.py`

```
usage: split_partner.py [-h] [-e EVALUE] [--linker_db LINKER_DB]
                        [--blast_path BLAST_PATH] [-o OUTPUT] [-t TRIM]
                        [-b BATCH] [-l LENGTH]
                        input type3_1 type3_2

DESCRIPTION: Run BLAST, find linker sequences and split two parts connected by
linkers

positional arguments:
  input                 the input fasta file containing fragment sequences of
```

```
                        type1 and type2
  type3_1               read_1 for evenlong (type3) fastq file
  type3_2               read_2 for evenlong (type3) fastq file

optional arguments:
  -h, --help            show this help message and exit
  -e EVALUE, --evalue EVALUE
                        cutoff evalues, only choose alignment with evalue less
                        than this cutoffs (default: 1e-5).
  --linker_db LINKER_DB
                        BLAST database of linker sequences
  --blast_path BLAST_PATH
                        path for the local blast program
  -o OUTPUT, --output OUTPUT
                        output file containing sequences of two sepatated
                        parts
  -t TRIM, --trim TRIM  trim off the first this number of nt as index,
                        default:10
  -b BATCH, --batch BATCH
                        batch this number of fragments for BLAST at a time.
                        default: 200000
  -r, --release         set to allow released criterion for Paired fragment in
                        Type 3, include those ones with no linker in two reads
  -l LENGTH, --length LENGTH
                        shortest length to be considered for each part of the
                        pair, default: 15

Library dependency: Bio, itertools
```

---

**Note:** New option added in version 0.3.1, which could allow two different strategies for selection of "Paired" fragments from the Type3 fragments. The `--release` option will allow a read pair to be called as "Paired" fragment even when the linker are not detected in both reads.

---

The linker fasta file contain sequences of all linkers

```
>L1
CTAGTAGCCCATGCAATGCGAGGA
>L2
AGGAGCGTAACGTACCCGATGATC
```

The output fasta files will be the input file name with different prefix ("NoLinker", "LinkerOnly", "BackOnly", "FrontOnly", "Paired") for different types. The other output file specified by `-o` contains information of aligned linker sequences for each Type1/2 fragment.

For example, if the commend is

```
split_partner.py fragment_ACCT.fasta evenlong_ACCTRm_dupPE_stitch_seq_1.fastq
    evenlong_ACCTRm_dupPE_stitch_seq_2.fastq
    -o fragment_ACCT_detail.txt --linker_db linker.fa
```

**Then, the output files will be:**

- backOnly_fragment_ACCT.fasta
- NoLinker_fragment_ACCT.fasta
- frontOnly_fragment_ACCT.fasta
- Paired1_fragment_ACCT.fasta

- Paired2_fragment_ACCT.fasta
- fragment_ACCT_detail.txt

The format of the last output file `fragment_ACCT_detail.txt` will be "Name | linker_num | linker_loc | Type | linker_order". Here are two examples:

```
HWI-ST1001:238:H0NYEADXX:1:1101:10221:1918      L1:2;L2:1  19,41;42,67;68,97       None    L2;L1;L1
HWI-ST1001:238:H0NYEADXX:1:1101:4620:2609       L1:2 28,46;47,79     Paired  L1;L1
```

In the **first** fragment, there are three regions can be aligned to linkers, 2 for L1 and 1 for L2, the order is L2, L1, L1. And they are aligned in region [19,41], [42,67], [68,97] of the fragment. "None" means this fragment is either 'LinkerOnly' or 'IndexOnly' (in this case it is 'LinkerOnly'). This fragment won't be written to any of the output fasta files.

In the **second** fragment, two regions can be aligned to linkers, and they are both aligned to L1. The two regions are in [28,46], [47,79] of the fragment. the fragment is "Paired" because on both two sides flanking the linker aligned regions, the length is larger than 15nt. The left part will be writen in `Paired1_fragment_ACCT.fasta` and the right part in `Paired2_fragment_ACCT.fasta`

### 2.2.5 Step 5: Align both parts of "Paired" fragment to the genome.

In this step, we will use the Paired1* and Paired2* fasta files output from the previous step. The sequences of part1 and part2 are aligned to the mouse genome mm9 with Bowtie and the pairs with both part1 and part2 mappable are selected as output. We also annotate the RNA types of each part in this step. All of these are implemented using script `Stitch-seq_Aligner.py`.

```
usage: Stitch-seq_Aligner.py [-h] [-s samtool_path] [-a ANNOTATION]
                             [-A DB_DETAIL]
                             miRNA_reads mRNA_reads bowtie_path miRNA_ref
                             mRNA_ref

Align miRNA-mRNA pairs for Stitch-seq. print the alignable miRNA-mRNA pairs
with coordinates

positional arguments:
  part1_reads           paired part1 fasta file
  part2_reads           paired part2 fasta file
  bowtie_path           path for the bowtie program
  part1_ref             reference genomic seq for part1
  part2_ref             reference genomic seq for part2

optional arguments:
  -h, --help            show this help message and exit
  -b, --bowtie2         set to use bowtie2 (--sensitive-local) for alignment,
                        need to change reference index and bowtie_path
  -u, --unique          set to only allow unique alignment
  -s samtool_path, --samtool_path samtool_path
                        path for the samtool program
  -a ANNOTATION, --annotation ANNOTATION
                        If specified, include the RNA type annotation for each
                        aligned pair, need to give bed annotation RNA file
  -A DB_DETAIL, --annotationGenebed DB_DETAIL
                        annotation bed12 file for lincRNA and mRNA with intron
                        and exon

Library dependency: Bio, pysam, itertools
```

An annotation file for different types of RNAs in mm9 genome (bed format, 'all_RNAs-rRNA_repeat.txt.gz') was included in Data folder. The annotation bed12 file for lincRNA and mRNA ('Ensembl_mm9.genebed.gz') was also included in Data folder. One can use the option `-a ../Data/all_RNAs-rRNA_repeat.txt.gz -A ../Data/Ensembl_mm9.genebed.gz` for annotation.

Here is a example:

```
Stitch-seq_Aligner.py Paired1_fragment_ACCT.fasta Paired2_fragment_ACCT.fasta
    ~/Software/bowtie-0.12.7/bowtie mm9 mm9 -s samtools
    -a ../Data/all_RNAs-rRNA_repeat.txt.gz -A ../Data/Ensembl_mm9.genebed.gz
    > ACCT_fragment_paired_align.txt
```

The format for the output file `ACCT_fragment_paired_align.txt` will be:

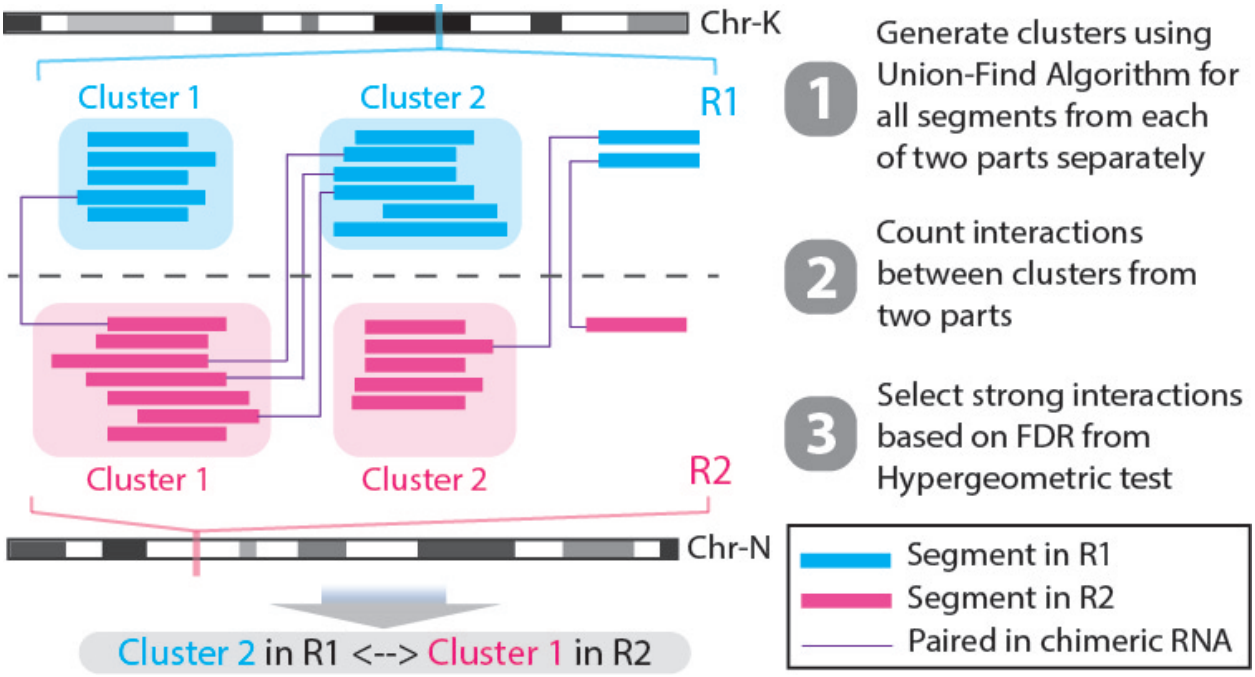
| Column [1] | Description |
|---|---|
| 1 | chromosome name of part1 |
| 2,3 | start/end position of part1 |
| 4 | strand information of part1 |
| 5 | sequence of part1 |
| 6 | RNA type for part1 |
| 7 | RNA name for part1 |
| 8 | RNA subtype [2] for part1 |
| 9 | name of the pair |

**Note:** Bowtie2 ("–sensitive-local" mode) option is added in version 0.3.1 for the user to choose, the `reference index` and `bowtie_path` need to be changed accordingly if you use bowtie2 instead of bowtie. User can also choose unique aligned reads or not by setting `--unique` option.

### 2.2.6 Step 6: Determine strong interactions.

In this step, we will generate clusters with high coverage separately for all part1 (R1) an part2 (R2) segments. Then based on the pairing information, we count the interactions between clusters from part1 and part2. The strong interactions can be selected by applying a p-value cutoff from hypergeometric test. (See figure below)

[1] column 10-17 are the same as column 1-8 except they are for part2 instead of part1.
[2] subtype can be intron/exon/utr5/utr3 for lincRNA and mRNA (protein-coding), '.' for others




We will use the script `Select_strongInteraction_pp.py`, parallel computing are implemented for clustering parallelly on different chromosomes:

```
usage: Select_strongInteraction_pp.py [-h] -i INPUT [-M MIN_CLUSTERS]
                                      [-m MIN_INTERACTION] [-p P_VALUE]
                                      [-o OUTPUT] [-P PARALLEL] [-F]

find strong interactions from paired genomic location data

optional arguments:
  -h, --help            show this help message and exit
  -i INPUT, --input INPUT
                        input file which is the output file of Stitch-seq-
                        Aligner.py
  -M MIN_CLUSTERS, --min_clusterS MIN_CLUSTERS
                        minimum number of segments allowed in each cluster,
                        default:5
  -m MIN_INTERACTION, --min_interaction MIN_INTERACTION
                        minimum number of interactions to support a strong
                        interaction, default:3
  -p P_VALUE, --p_value P_VALUE
                        the p-value based on hypergeometric distribution to
                        call strong interactions, default: 0.05
  -o OUTPUT, --output OUTPUT
                        specify output file
  -P PARALLEL, --parallel PARALLEL
                        number of workers for parallel computing, default: 5
  -F, --FDR             Compute FDR if specified

need Scipy for hypergeometric distribution
```

The input of the script is the output of Step 5 (`ACCT_fragment_paired_align.txt` in the example). "annotated_bed" class is utilized in this script.

Here is a example:

```
Select_strongInteraction.py -i ACCT_fragment_paired_align.txt -o ACCT_interaction_clusters.txt
```

The column description for output file `ACCT_interaction_clusters.txt` is:

| Column | Description |
|---|---|
| 1 | chromosome name of cluster in part1 |
| 2,3 | start/end position of cluster in part1 |
| 4 | RNA type for cluster in part1 |
| 5 | RNA name for cluster in part1 |
| 6 | RNA subtype for cluster in part1 |
| 7 | # of counts for cluster in part1 |
| 8-14 | Same as 1-7, but for cluster in part2 |
| 15 | # of interactions between these two clusters |
| 16 | log(p-value) of the hypergeometric testing |

### 2.2.7 Step 7: Visualization of interactions and coverages.

There are two ways of visulization provided ( LOCAL and GLOBAL ):

- *Visualization of local interactions*.
- *Visualization of global interactome*.

## 2.3 Other functions

### 2.3.1 Determine the RNA types of different parts within fragments.

### 2.3.2 Find linker sequences within the library.

### 2.3.3 Find intersections between two different interaction sets based on genomic locations

The script tool `intersectInteraction.py` could be used to identify overlap of interactions between two interaction set from independent experiments based on genomic locations (two replicates or two different samples)

```
usage: intersectInteraction.py [-h] -a FILEA -b FILEB [-s START] [-n NBASE]
                               [-o OUTPUT] [-c]

find intersections (overlaps) between two interaction sets

optional arguments:
  -h, --help            show this help message and exit
  -a FILEA, --filea FILEA
                        file for interaction set a
  -b FILEB, --fileb FILEB
                        file for interaction set b
  -s START, --start START
                        start column number of the second part in each
                        interaction (0-based), default:7
  -n NBASE, --nbase NBASE
                        number of overlapped nucleotides for each part of
                      interactions to call intersections, default: 1
  -o OUTPUT, --output OUTPUT
```

```
                        specify output file
  -p, --pvalue          calculate p-values based on 100times permutations

require 'random'&'numpy'&'scipy' module if set '-p'
```

if "-p" option is set, then the program will do permutation for 100 times by shuffling the two partners of interactions in set a. A p-value will be calculate based on permutation distribution.

### 2.3.4 Find intersections between two different interaction sets based on annotation

The script tool `intersectInteraction_genePair.R` could be used to identify overlap of interactions between two interaction set from independent experiments based on the RNA annotations (two replicates or two different samples)

```
usage: intersectInteraction_genePair.R [-h] [-n NUM [NUM ...]] [-p] [-r]
                                       [-o OUTPUT]
                                       interactionA interactionB

Call intersections based on gene pairs

positional arguments:
  interactionA          the interaction file a,[required]
  interactionB          the interaction file b,[required]

optional arguments:
  -h, --help            show this help message and exit
  -n NUM [NUM ...], --num NUM [NUM ...]
                        Column numbers for the gene name in two part,[default:
                        [5, 12]]
  -p, --pvalue          set to do 100 permutations for p-value of overlap
  -r, --release         set to only require match of chromosome and RNA name,
                        but not subtype
  -o OUTPUT, --output OUTPUT
                        output intersection file name, pairs in A that overlap
                        with B, [default: intersect.txt]
```

if "-p" option is set, then the program will do permutation for 100 times by shuffling the two partners of interactions in both set a and set b. A p-value will be calculate based on permutation distribution.

### 2.3.5 RNA structure prediction by adding digestion site information

The script will take selfligated chimeric fragments from given snoRNA (ID) and predict secondary structures with and without constraints of digested single strand sites. It is also able to compare the known structure in dot format if the known structure is available and specified by "-a". The script needs RNAStructure software for structure prediction ("-R") and and VARNA command line tool for visualization ("-v").

```
usage: RNA_structure_prediction.py [-h] [-g GENOMEFA] [-R RNASTRUCTUREEXE]
                                   [-a ACCEPTDOT] [-o OUTPUT]
                                   [-s samtool_path] [-v VARNA]
                                   [-c COLORMAPSTYLE]
                                   ID linkedPair

plot RNA structure with distribution of digested end, refine structure with
loc of digested end

positional arguments:
```

```
  ID                    Ensembl gene ID of RNA
  linkedPair            file for information of linked pairs, which is output
                        of 'Stitch-seq_Aligner.py'

optional arguments:
  -h, --help            show this help message and exit
  -g GENOMEFA, --genomeFa GENOMEFA
                        genomic sequence,need to be fadix-ed
  -R RNASTRUCTUREEXE, --RNAstructureExe RNASTRUCTUREEXE
                        folder of RNAstrucutre suite excutable
  -a ACCEPTDOT, --acceptDot ACCEPTDOT
                        accepted structure in dot format, for comparing of
                        accuracy, no comparison if not set
  -o OUTPUT, --output OUTPUT
                        output distribution of digested sites with dot
                        structures, can be format of eps, pdf, png,...
  -s samtool_path, --samtool_path samtool_path
                        path for the samtool program
  -v VARNA, --varna VARNA
                        path for the VARNA visualization for RNA
  -c COLORMAPSTYLE, --colorMapStyle COLORMAPSTYLE
                        style of color map, choose from: "red", "blue",
                        "green", "heat", "energy", and "bw",default:"heat"
```

Here is a example:

```
python RNA_structure_prediction.py \
  ENSMUSG00000064380 \
  /data2/sysbio/UCSD-sequencing/2013-11-27-Bharat_Tri_Shu/Undetermined_indices/Sample_lane8/ACCT_GGC
  -a Snora73_real_dot.txt \
  -o Snora73_distribution.pdf
```

Here "Snora73_real_dot.txt" is dot format of known Snora73 structure This will generate three eps files with secondary structures ("Predict", "Refine", "Accepted (known)". Also the output pdf file contains the distribution of digested sites in whole RNA molecule.

CHAPTER

THREE

# VISUALIZATION OF LOCAL RNA-RNA INTERACTIONS

## 3.1 Prerequirement

This program require python modules: xplib, matplotlib, numpy, bx-python

## 3.2 Run the program to generate visualization

The script "Plot_interaction.py" will be used for this purpose,

```
usage: Plot_interaction.py [-h] [-n N] [-s START [START ...]] [-d DISTANCE]
                           [-g GENEBED] [-w PHYLOP_WIG] [-p PAIR_DIST] [-S]
                           [-o OUTPUT]
                           interaction linkedPair

plot linked pairs around a given interaction. information of linked pairs are
stored in file '*_fragment_paired_align.txt'

positional arguments:
  interaction           Interaction file from output of
                        'Select_strongInteraction_pp.py'
  linkedPair            file for information of linked pairs, which is output
                        of 'Stitch-seq_Aligner.py'

optional arguments:
  -h, --help            show this help message and exit
  -n N                  Choose region to plot, it can be a number (around n-th
                        interaction in the interaction file). This is mutually
                        exclusive with '-r' option
  -r R [R ...]          Choose region to plot, give two interaction regions
                        with format 'chr:start-end', this is mutually
                        exclusive with '-n' option
  -s START [START ...], --start START [START ...]
                        start column number of the second region in
                        interaction file and linkedPair file, default=(7,8)
  -d DISTANCE, --distance DISTANCE
                        the plus-minus distance (unit: kbp) flanking the
                        interaction regions to be plotted, default=10
  -g GENEBED, --genebed GENEBED
                        the genebed file from Ensembl, default:
                        ../Data/Ensembl_mm9.genebed
  -w PHYLOP_WIG, --phyloP_wig PHYLOP_WIG
                        the bigWig file for phyloP scores,defualt:
```

```
                        mouse.phyloP30way.bw
  -p PAIR_DIST, --pair_dist PAIR_DIST
                        two interacted parts within this distance are
                        considered as self-ligated and they are marked or
                        eliminated (see option -s for slim mode), default:
                        200bp
  -S, --Slim            set slim mode to eliminate self ligated interactions
  -o OUTPUT, --output OUTPUT
                        output plot file, can be format of emf, eps, pdf, png,
                        ps, raw, rgba, svg, svgz
```

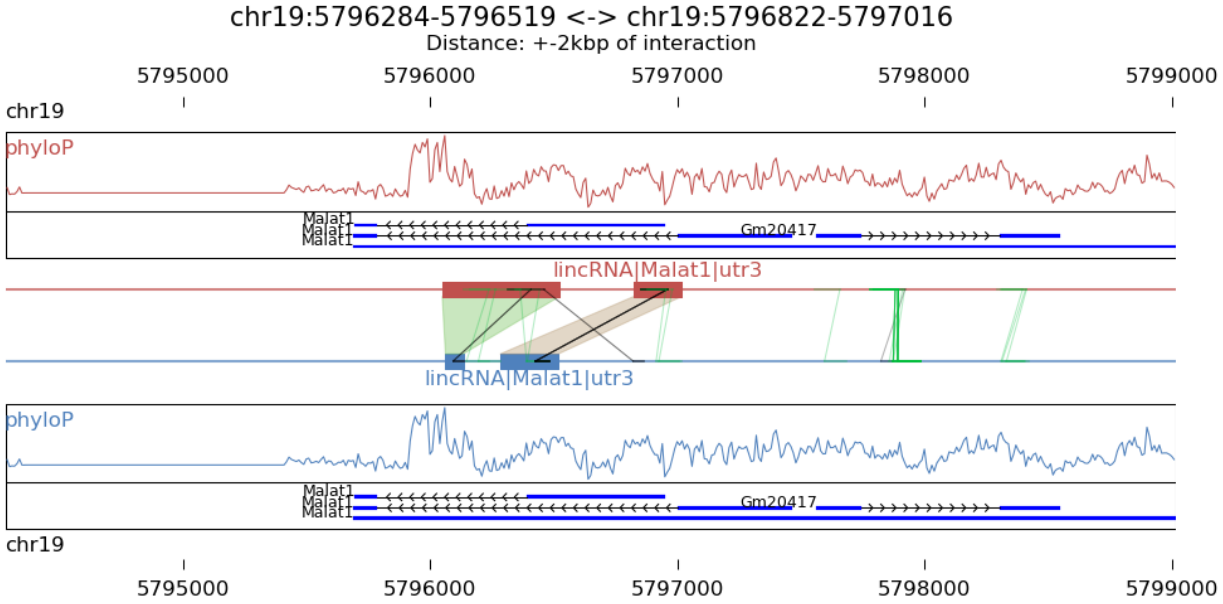
**Note:** linkedPair file is the output *_fragment_paired_align.txt from *Step5:Stitch-seq_Aligner.py* of the pipeline; Interaction txt file is the output of *Step6:Select_strongInteraction_pp.py*.

## 3.3 Example of result graph

*Example code:*

```
python Plot_interaction.py
        ACCT_interaction_clusters_rmrRNA.txt \
        ACCT_fragment_paired_align_rmRNA_sort.txt.gz \
        -n 2412 \
        -d 5 \
        -o local_interaction.pdf
```

*Result figure:*




*Explanation:*

CHAPTER

FOUR

# VISUALIZATION OF GLOBAL RNA-RNA INTERACTOME

## 4.1 Prerequirement

This program is powered by RCircos.

Required R packages (our program will check for the presence of these packages and install/load them automatically if not present):

- argparse, RCircos, biovizBase, rtracklayer

The program also require a python script "bam2tab.py" (already in /bin/ folder) to call coverage from BAM2X

## 4.2 Run the program to generate visualization

We will use the script "Plot_Circos.R" for this purpose.

```
usage: Plot_Circos.R [-h] [-g GENOME] [-b BIN] [-o OUTPUT]
                     interaction part1 part2

positional arguments:
  interaction           the interaction file,[required]
  part1                 aligned BAM file for part1,[required]
  part2                 aligned BAM file for part2,[required]

optional arguments:
  -h, --help            show this help message and exit
  -g GENOME, --genome GENOME
                        genome information, choice: mm9/mm10/hg19 et.al.,
                        [default: mm9]
  -b BIN, --bin BIN     window size for the bins for coverage calling, [default: 100000.0]
  -o OUTPUT, --output OUTPUT
                        output pdf file name, [default: Interactome_view.pdf]
```

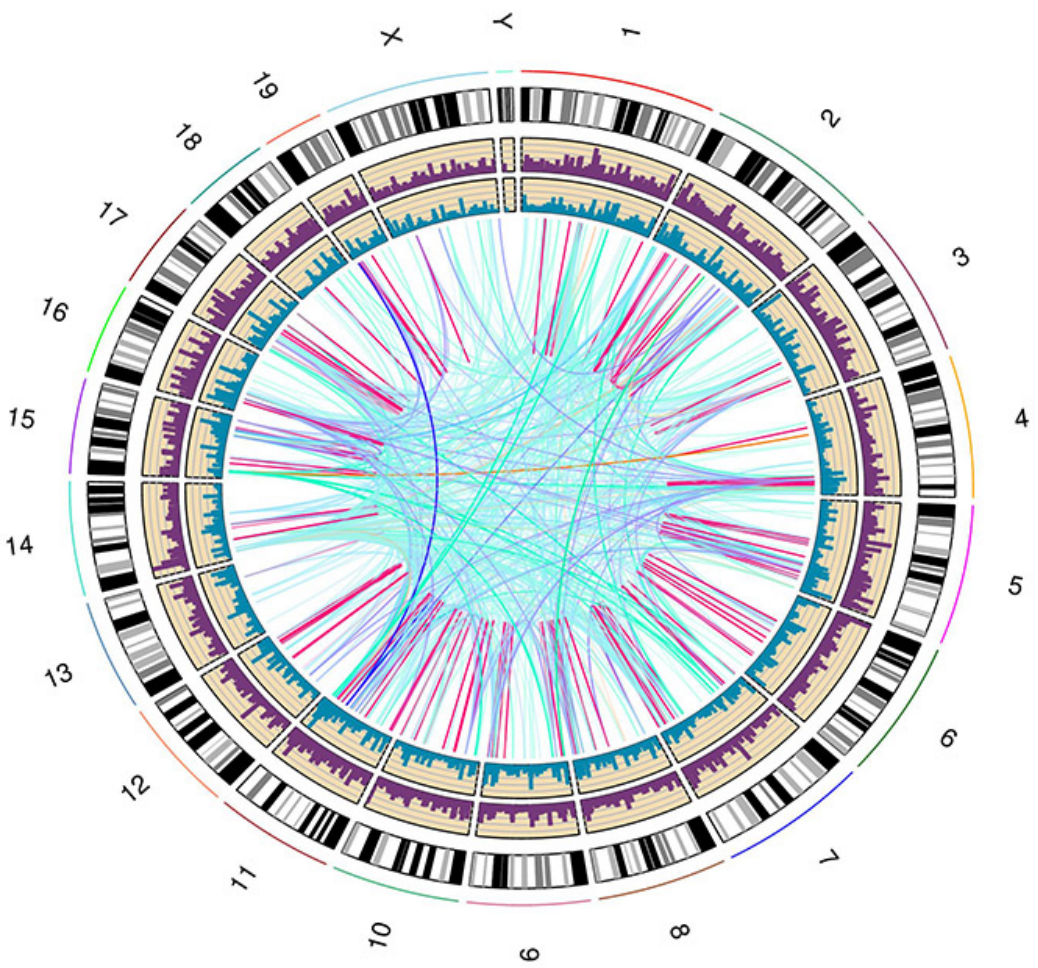
**Note:** part1, part2 BAM files are the ones generated from *Step5:Stitch-seq_Aligner.py* of the pipeline; Interaction txt file is the output of *Step6:Select_strongInteraction_pp.py*.

## 4.3 Example of result graph

*Example code:*

```
Rscript Plot_Circos.R GGCG_interaction_clusters.txt
  sort_Paired1_fragment_GGCG.bam sort_Paired2_fragment_GGCG.bam
  -b 100000 -o Interactome_GGCG.pdf
```

*Result figure:*




*Explanation:*

CHAPTER

FIVE

# VISUALIZATION OF INTERACTION TYPES ENRICHMENT

## 5.1 Prerequirement

Required R packages (our program will check for the presence of these packages and install/load them automatically if not present):

- "argparse","ggplot2","scales"

## 5.2 Run the program to generate visualization for enrichment of different types of interactions

We will use the script "Interaction_type_enrichment.R" for this purpose.

```
usage: ../../bin/Interaction_type_enrichment.R [-h] [-n NUM [NUM ...]]
                                               [-o OUTPUT]
                                               interaction

 plot the statistical significance for enrichment of different interaction
 types

 positional arguments:
   interaction           the strong interaction file,[required]

 optional arguments:
   -h, --help            show this help message and exit
   -n NUM [NUM ...], --num NUM [NUM ...]
                         Column numbers for the type name in two part,[default:
                         [4, 11]]
   -o OUTPUT, --output OUTPUT
                         output pdf figure file, [default:
                         interaction_type.pdf]
```

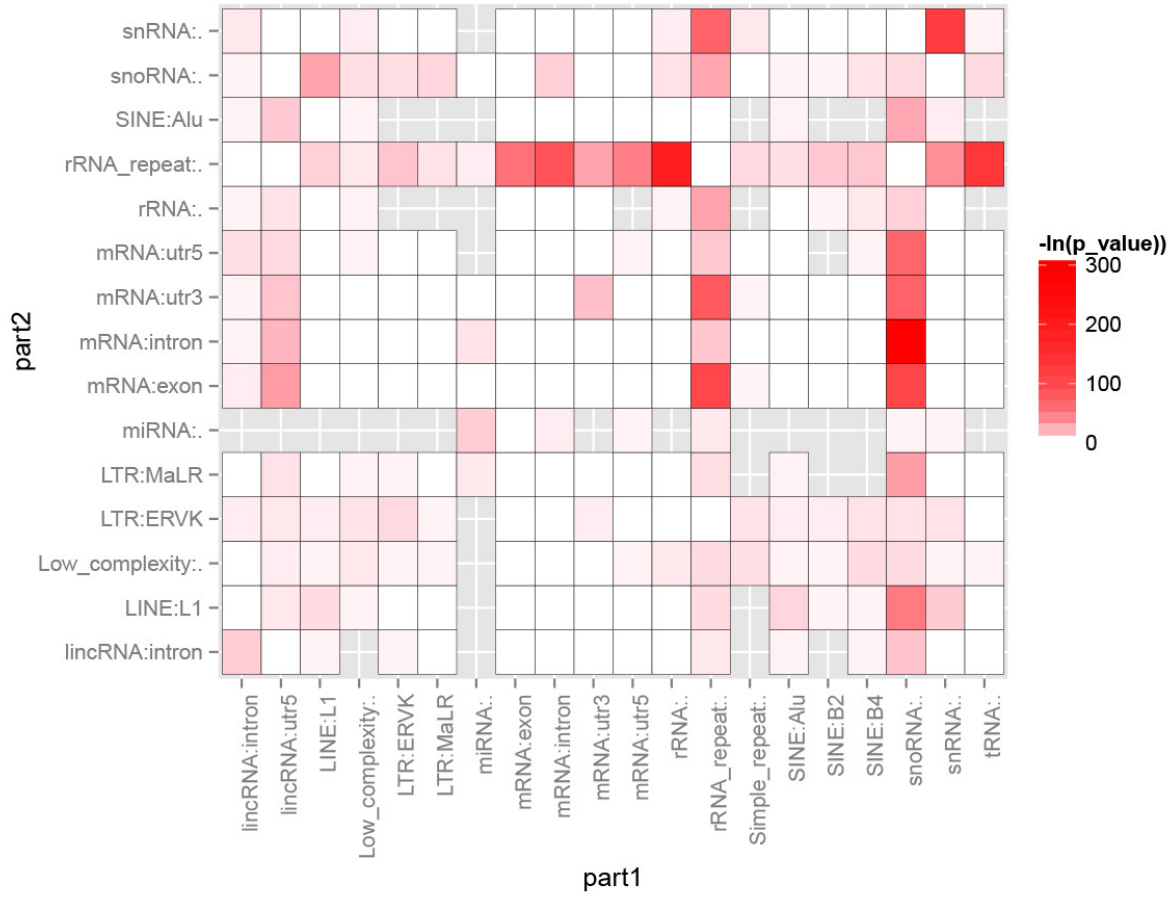
**Note:** Interaction txt file is the output of *Step6:Select_strongInteraction_pp.py*.

## 5.3 Example of result graph

*Example code:*

```
Rscript Plot_Circos.R ACCT_interaction_clusters.txt
  -n 4 11 -o ACCT_interaction_type.pdf
```

*Result figure:*




*Explanation:*

For each interaction types (Type1_in_Part1<->Type2_in_Part2), we calculated the number of Type1 in Part1 from all intereactions `n1` and number of Type2 in Part2 from all interactions `n2`. Then we calculate the number of interactions with this type: Type1_in_Part1<->Type2_in_Part2 `n12`. The p-value for each interacction type is calculated based on the hypergeometric distribution with R command: `phyper(n12, n1, total_n - n1, n2, lower.tail=F)`. Here `total_n` is the total number of strong interactions. The color for each cell (each interaction type) are coded based on the value of "-ln(p-value)".

CHAPTER

SIX

# PYTHON APIS CREATED FOR THIS PROJECT

## 6.1 Annotation module

For the purpose of annotating RNA types for genomic regions.

Annotation.**overlap**(*bed1*, *bed2*)

This function compares overlap of two Bed object from same chromosome

**Parameters**

- **bed1** – A Bed object from xplib.Annotation.Bed (BAM2X)
- **bed2** – A Bed object from xplib.Annotation.Bed (BAM2X)

**Returns** boolean – True or False

Example:

```
>>> from xplib.Annotation import Bed
>>> from Annotation import overlap
>>> bed1=Bed(["chr1",10000,12000])
>>> bed2=Bed(["chr1",9000,13000])
>>> print overlap(bed1,bed2)
True
```

Annotation.**Subtype**(*bed1*, *genebed*)

This function determines intron or exon or utr from a BED12 file.

**Parameters**

- **bed1** – A Bed object defined by xplib.Annotation.Bed (BAM2X)
- **genebed** – A Bed12 object representing a transcript defined by xplib Annotaton.Bed with information of exon/intron/utr from an BED12 file

**Returns** str – RNA subtype. “intron”/”exon”/”utr3”/”utr5”/”.”

Example:

```
>>> from xplib.Annotation import Bed
>>> from xplib import DBI
>>> from Annotation import Subtype
>>> bed1=Bed(["chr13",40975747,40975770])
>>> a=DBI.init("../../Data/Ensembl_mm9.genebed.gz","bed")
>>> genebed=a.query(bed1).next()
>>> print Subtype(bed1,genebed)
"intron"
```

Annotation.**annotation**(*bed*, *ref_allRNA*, *ref_detail*, *ref_repeat*)

This function is based on `overlap()` and `Subtype()` functions to annotate RNA type/name/subtype for any genomic region.

**Parameters**

- **bed** – A Bed object defined by xplib.Annotation.Bed (in BAM2X).
- **ref_allRNA** – the DBI.init object (from BAM2X) for bed6 file of all kinds of RNA
- **ref_detail** – the DBI.init object for bed12 file of lincRNA and mRNA with intron, exon, UTR
- **ref_detail** – the DBI.init object for bed6 file of mouse repeat

**Returns** list of str – [type,name,subtype]

Example:

```
>>> from xplib.Annotation import Bed
>>> from xplib import DBI
>>> from Annotation import annotation
>>> bed=Bed(["chr13",40975747,40975770])
>>> ref_allRNA=DBI.init("../../Data/all_RNAs-rRNA_repeat.txt.gz","bed")
>>> ref_detail=DBI.init("../../Data/Ensembl_mm9.genebed.gz","bed")
>>> ref_repeat=DBI.init("../../Data/mouse.repeat.txt.gz","bed")
>>> print annotation(bed,ref_allRNA,ref_detail,ref_repeat)
["protein_coding","gcnt2","intron"]
```

## 6.2 "annotated_bed" data class

**class** data_structure.**annotated_bed**(*x=None*, ***kwargs*)

To store, compare, cluster for the genomic regions with RNA annotation information. Utlized in the program *Select_stronginteraction_pp.py*

**Cluster**(*c*)

Store cluster information of self object

**Parameters** **c** – cluster index

Example:

```
>>> a=annotated_bed(chr="chr13",start=40975747,end=40975770)
>>> a.Cluster(3)
>>> print a.cluster
3
```

**Note:** a.cluster will be the count information when a become a cluster object in *Select_stronginteraction_pp.py*

**Update**(*S*, *E*)

Update the upper and lower bound of the cluster after adding segments using Union-Find.

**Parameters**

- **S** – start loc of the newly added genomic segment
- **E** – end loc of the newly added genomic segment

Example:

```
>>> a=annotated_bed(chr="chr13",start=40975747,end=40975770)
>>> a.Update(40975700,40975800)
>>> print a.start, a.end
40975700 40975800
```

**__init__**(*x=None*, ***kwargs*)

Initiation example:

```
>>> str="chr13  40975747        40975770        +       ATTAAG...TGA    protein_coding  gcnt
>>> a=annotated_bed(str)
or
>>> a=annotated_bed(chr="chr13",start=40975747,end=40975770,strand='+',type="protein_coding"
```

**__lt__**(*other*)

Compare two objects self and other when they are not **overlapped**

**Parameters** **other** – another `annotated_bed` object

**Returns** boolean – "None" if overlapped.

Example:

```
>>> a=annotated_bed(chr="chr13",start=40975747,end=40975770)
>>> b=annotated_bed(chr="chr13",start=10003212,end=10005400)
>>> print a>b
False
```

**__str__**()

Use print function to output the cluster information (chr, start, end, type, name, subtype,cluster)

Example:

```
>>> str="chr13  40975747        40975770        +       ATTAAG...TGA    protein_coding  gcnt
>>> a=annotated_bed(str)
>>> a.Cluster(3)
>>> a.Update(40975700,40975800)
>>> print a
"chr13  40975700        40975800        protein_coding  gcnt2   intron  3"
```

**overlap**(*other*)

Find overlap between regions

**Parameters** **other** – another `annotated_bed` object

**Returns** boolean

## 6.3 "RNAstructure" class

**class** `RNAstructure.`**RNAstructure**(*exe_path=None*)

Interface class for RNAstructure executable programs.

**DuplexFold**(*seq1=None*, *seq2=None*, *dna=False*)

Use "DuplexFold" program to calculate the minimum folding between two input sequences

**Parameters**

- **seq1,seq2** – two DNA/RNA sequences as string, or existing fasta file name
- **dna** – boolean input. Specify then DNA parameters are to be used

**Returns** minimum binding energy, (unit: kCal/Mol)

Example:

```
>>> from RNAstructure import RNAstructure
>>> RNA_prog = RNAstructure(exe_path="/home/yu68/Software/RNAstructure/exe/")
>>> seq1 = "TAGACTGATCAGTAAGTCGGTA"
>>> seq2 = "GACTAGCTTAGGTAGGATAGTCAGTA"
>>> energy=RNA_prog.DuplexFold(seq1,seq2)
>>> print energy
```

**Fold**(*seq=None*, *ct_name=None*, *sso_file=None*, *Num=1*)

Use "Fold" program to predict the secondary structure and output dot format.

**Parameters**

- **seq** – one DNA/RNA sequence as string, or existing fasta file name
- **ct_name** – specify to output a ct file with this name, otherwise store in temp, default: None
- **sso_file** – give a single strand offset file, format see http://rna.urmc.rochester.edu/Text/File_Formats.html#Offset
- **Num** – choose Num th predicted structure

**Returns** dot format of RNA secondary structure and RNA sequence.

Example:

```
>>> from RNAstructure import RNAstructure
>>> RNA_prog = RNAstructure(exe_path="/home/yu68/Software/RNAstructure/exe/")
>>> seq = "AUAUAAUUAAAAAAUGCAACUACAAGUUCCGUGUUUCUGACUGUUAGUUAUUGAGUUAUU"
>>> sequence,dot=RNA_prog.Fold(seq)
>>> assert(seq==sequence)
```

**__init__**(*exe_path=None*)

Initiation of object

**Parameters** **exe_path** – the folder path of the RNAstructure executables

Example:

```
>>> from RNAstructure import RNAstructure
>>> RNA_prog = RNAstructure(exe_path="/home/yu68/Software/RNAstructure/exe/")
```

**scorer**(*ct_name1*, *ct_name2*)

Use 'scorer' pogram to compare a predicted secondary structure to an accepted structure. It calculates two quality metrics, sensitivity and PPV

**Parameters**

- **ct_name1** – The name of a CT file containing predicted structure data.
- **ct_name2** – The name of a CT file containing accepted structure data, can only store one structure.

**Returns** sensitivity, PPV, number of the best predicted structure.

Example:

```
>>> ct_name1 = "temp_prediction.ct"
>>> ct_name2 = "temp_accept.ct"
>>> from RNAstructure import RNAstructure
>>> RNA_prog = RNAstructure(exe_path="/home/yu68/Software/RNAstructure/exe/")
>>> sensitivity, PPV, Number = RNA_prog.scorer(ct_name1,ct_name2)
```

CHAPTER

SEVEN

# RESOURCES OF STRONG INTERACTIONS FROM TWO MOUSE CELL TYPES

## 7.1 Description of different samples

### 7.1.1 E14_WP_1

**Cell line** ESC E14

**Barcode** ACCT

**Experimental Details** Actively growing E14 cells were UV irradiated (254 nm) at 200mJ/cm to crosslink proteins to interacting RNAs. After cell lysis, we trim down RNAs into 1000-2000 nt using RNase I and remove DNA by TURBO DNase. To recover RNAs bound to RNA-binding proteins, we biotin-labeled them with EZ-Link Iodoacetyl-PEG2-Biotin from Pierce. RNA-protein complexes were next immobilized on Streptavidin-coated beads. The beads are then saturated with free biotin, preventing it from interfering with following ligation with biotin-tagged linker. A biotin-tagged RNA linker was ligated to the 5'-end of immobilized RNAs. Proximity ligation was then carried out under diluted conditions while the RNA-protein complexes are still bound on bead. After RNA purification by Proteinase K and phenol-chloroform extraction, we specifically removed the unligated biotin by first anneal a complementary DNA oligo to the biotin-tagged linker by using the annealing protocol: 70oC for 5 min, 25oC for 20 min, T7 Exo for 30 min. Exonuclease T7 was added to remove terminal unligated biotin at the double-stranded RNAlinker-DNAoligo hybrid. T7 Exonuclease acts in the 5' to 3' direction, catalyzing the removal of 5' mononucleotides from duplex DNA and RNA/DNA hybrids in the 5' to 3' direction. The resulted RNAs were fragmented again into ~200 nt using RNase III RNA Fragmentation Module from NEB (1ul of RNase III in 6 min at 37C). The RNAs were purified by column and ligated with sequencing adapter, then reverse-transcribed and PCR for library construction. We applied an rRNA removal step after constructing cDNA by using an rRNA removal protocol based on the Duplex-Specific thermostable nuclease (DSN) enzyme using the protocol recommended by Illumina. The constructed cDNAs were quality-checked by Bioanalyzer. The cDNAs were next subjected paired-end sequencing on HiSeq-2500 platform.

**Linker** *mL5*: 5' - rCrUrA rG/iBiodT/rA rGrCrC rCrArU rGrCrA rArUrG rCrGrA rGrGrA - 3'

### 7.1.2 E14_WP_2

**Cell line** ESC E14

**Barcode** GGCG

**Experimental Details** Same as E14_WP_1 but this time rRNA removal was performed right after Proteinase K and phenol-chloroform treatment using the GeneRead rRNA Depletion Kit by Qiagen.

Furthermore, the annealing of RNA linker and complementary DNA oligo was changed into: denature for 90 s at 90°C, and then anneal at -0.1°C/s to 25 °C and then incubate for 25 min at 25 °C. Since after rRNA depletion the amount of RNA remained was less than that obtained from E14 #1, we reduced the duration of RNA fragmentation by RNase III from 6 min to 3 min. However, this reduction in RNase III treatment led to large fragments than desirable.

**Linker** *mL5*: 5' - rCrUrA rG/iBiodT/rA rGrCrC rCrArU rGrCrA rArUrG rCrGrA rGrGrA - 3'

### 7.1.3 E14_WP_3

**Cell line** ESC E14

**Barcode** AATG

**Experimental Details** To detect interactions between RNAs that are not bound to the same protein but to interacting proteins, we used formaldehyde in conjunction with a second crosslinker, EGS. The combination of formaldehyde and EGS crosslinks both RNA-protein and protein-protein interactions thereby maximize the detection of RNA-RNA interactions that are facilitated by interacting proteins. Actively grown E14 cells was crosslinked with 1.5 mM of freshly prepared EthylGlycol bis(SuccinimidylSuccinate) (EGS)) for 45 minutes at room temperature and then 1% of formaldehyde for 10 minutes also at room temperature. Since crosslinking by formaldehyde makes the cells very rigid and less amenable to be broken down lysis buffer. Therefore, we utilized sonication to fragment the protein-bound RNA into ~1000 nt size range. The remaining steps were performed similarly to E14_WP_2.

Another main difference between this sample and other samples is that we didn't remove the nuclei, thus effectively including RNA-RNA interactions inside the nucleus into the sample. In other samples, only the cytoplasm was enriched.

**Linker** *mL5*: 5' - rCrUrA rG/iBiodT/rA rGrCrC rCrArU rGrCrA rArUrG rCrGrA rGrGrA - 3'

### 7.1.4 MEF_WP_1

**Cell line** ESC E14

**Barcode** GGCG

**Experimental Details** We irradiated actively grown 1E8 MEF cells (254 nm). This time, the RNAs were fragmented into 300nt size range. RNase III fragmentation was also modified accordingly to adjust for smaller amount of RNAs: instead of adding 1ul of RNase III, we added only 0.5uL of RNase III and then incubated at 37C for 5 min. The subsequent steps were performed using the same procedure as E14_WP_2.

**Linker** *mL5*: 5' - rCrUrA rG/iBiodT/rA rGrCrC rCrArU rGrCrA rArUrG rCrGrA rGrGrA - 3'

## 7.2 Resources of Strong Interactions

### 7.2.1 From merged data of E14_WP_1 and E14_WP_2:

Download

### 7.2.2 From E14_WP_3 dual crosslinking:

Download

### 7.2.3 From MEF_WP_1 sample:

Download

---

**Note:** RNA Hi-C tools benifits a lot from BAM2X, a convenient python interface for most common NGS datatypes. Try BAM2X now!

---

CHAPTER

# EIGHT

# UPDATES

**2014-5-15:**

- Add result *resources* for identified strong interactions in mouse E14 cells and MEF cells.

**2014-05-14:**

- Add new function to find overlap between two interaction sets based on their RNA annotations, see: *intersectInteraction_genePair.R*.
- Allow input of two genomic regions to visualize local interactions using `-r` option in "*Plot_interaction.py*" function

**2014-05-11:**

- Add new function to show enrichment of different types of interactions: *Interaction_type_enrichment.R*.

**Version 0.3.2 (2014-05-07):**

- change the name into RNA-Hi-C

**2014-05-06:**

- In "*Select_strongInteraction_pp.py*" function, now annotations are updated after doing clustering and for strong interaction. The indexing of annotation files may take some time.
- New "*RNA_structure_prediction.py*" function to refine RNA structure prediction based on empirical offset of free energies for single strand nucleotide.

**New features in 0.3.1 (2014-05-02):**

- Add "–release" option in "*split_partner.py*" function. Allow a Type3 read-pair considered to be a "Paired" chimeric fragment even linker does not show up.
- Fix bugs in "*Select_strongInteraction_pp.py*" function when the number of mapped pairs is low and some chromosomes don't have any mapped read in part1 or part2.
- Add bowtie 2 option and Unique-align option in "*Stitch-seq_Aligner.py*" function.
- Different colors for different types of interactions in the *visualization of interactome*.
- New API for folding energies of two RNA molecules, see "*RNAstructure*".
- Allow permutation-based strategies to calculate the p-value for the overlap between two independent interaction sets in "*intersectInteraction.py*" function

**New features in 0.2.2:**

- "*Plot_interaction.py*" function to plot local RNA-RNA interactions.
- "*intersectInteraction.py*" function to call overlap between two independent interaction sets.

CHAPTER

NINE

# INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

## a

## Symbols



## A



## C



## D



## F



## I



## O



## P



## R



## S



## U