Stitch-seq-tools Documentation

Release 1.0.1

Pengfei Yu

September 27, 2013

CONTENTS

Contents:

CHAPTER

ONE

STITCH-SEQ-TOOLS 1.0 DOCUMENTATION

1.1 Installation

1.1.1 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`)
- PyCogent (for annotation of RNA types) [see note]

2. The Boost.Python C++ library

Note: the Annotation feature need the development version of PyCogent (install instruction). Since we need the getTranscriptByStableId function which is described here.

1.1.2 Step 2: Download the package

Clone the package from GitHub:

```
git clone http://github.com/yu68/stitch-seq.git
```

1.1.3 Step 3: Add library source to your python path

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

```
Location="/path/of/Stitch-seq-tools" # change accordingly
export PYTHONPATH="$Location/src:$PYTHONPATH"
export PATH="$PATH:$Location/bin"
```

1.2 Overview

Stitch-seq-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.

1.3 Support

For issues related to the use of Stitch-seq-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.*

Other functions:

1. *Determine the RNA types of different parts within fragments.*
2. *Find linker sequences within the library.*

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:

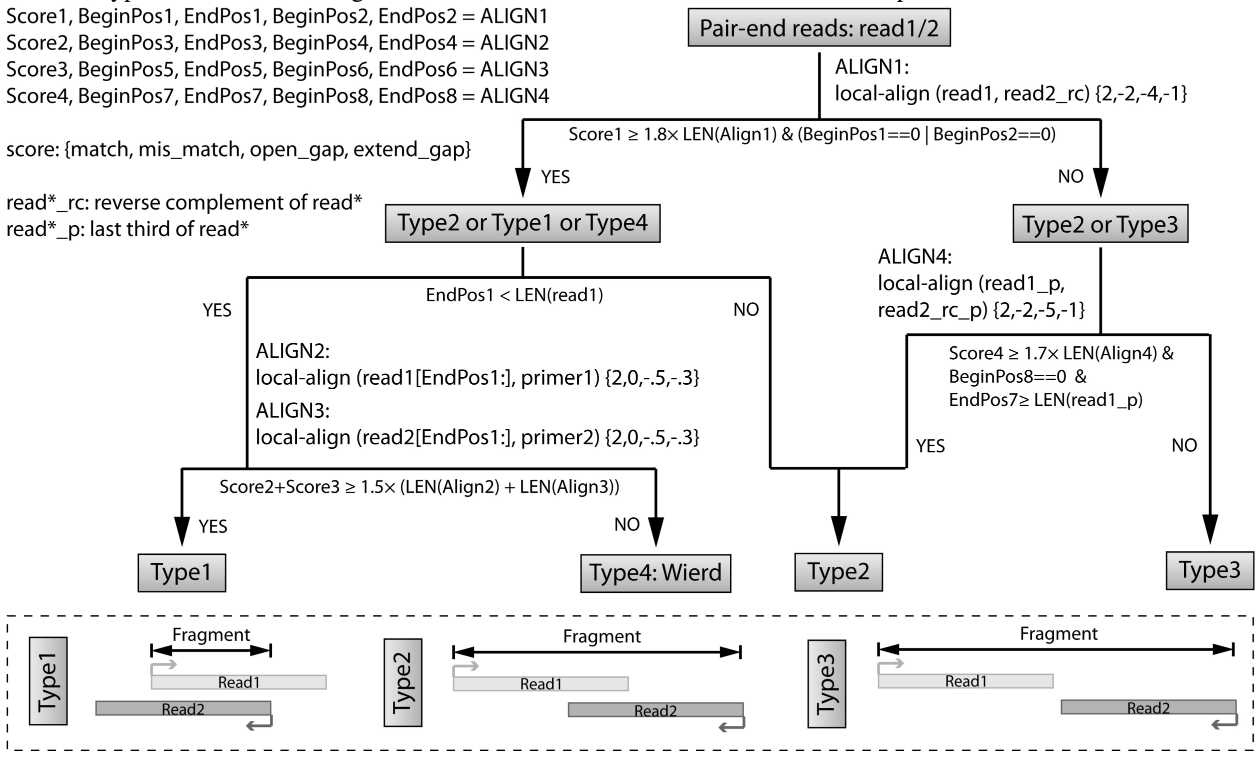
- ACCT_Rm_dupPE_example.F1.fastq

- 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 3-6) are 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.

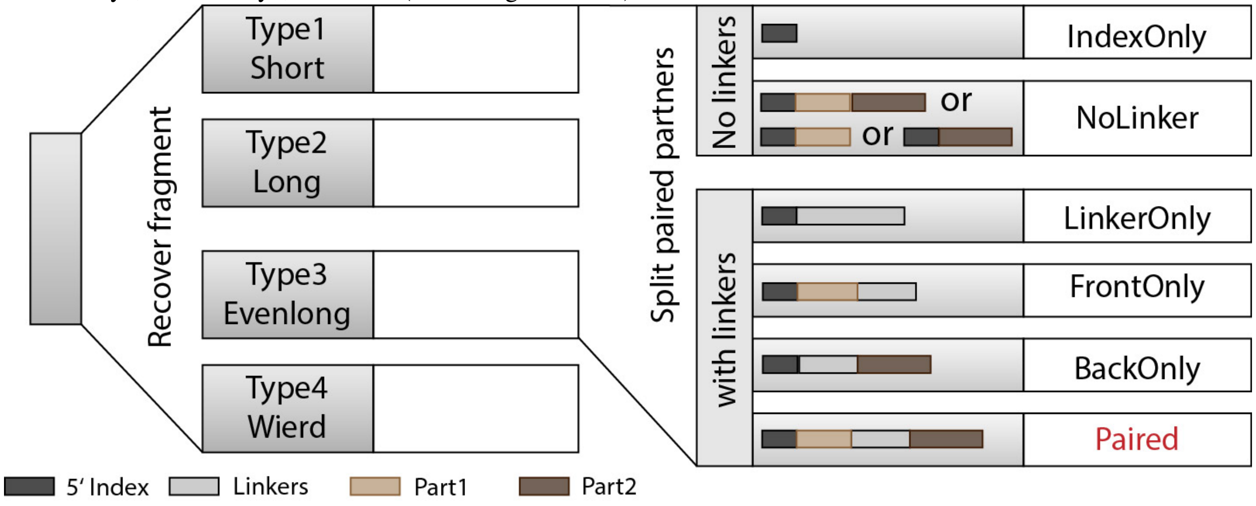

Score1, BeginPos1, EndPos1, BeginPos2, EndPos2 = ALIGN1
Score2, BeginPos3, EndPos3, BeginPos4, EndPos4 = ALIGN2
Score3, BeginPos5, EndPos5, BeginPos6, EndPos6 = ALIGN3
Score4, BeginPos7, EndPos7, BeginPos8, EndPos8 = ALIGN4
score: {match, mis_match, open_gap, extend_gap}
read*_rc: reverse complement of read*
read*_p: last third of read*
Pair-end reads: read1/2
ALIGN1:
local-align (read1, read2_rc) {2,-2,-4,-1}
Score1 ≥ 1.8× LEN(Align1) & (BeginPos1==0 | BeginPos2==0)
YES
NO
Type2 or Type1 or Type4
Type2 or Type3
ALIGN4:
local-align (read1_p, read2_rc_p) {2,-2,-5,-1}
EndPos1 < LEN(read1)
YES
NO
ALIGN2:
local-align (read1[EndPos1:], primer1) {2,0,-.5,-.3}
ALIGN3:
local-align (read2[EndPos1:], primer2) {2,0,-.5,-.3}
Score4 ≥ 1.7× LEN(Align4) & BeginPos8==0 & EndPos7≥ LEN(read1_p)
YES
NO
Score2+Score3 ≥ 1.5× (LEN(Align2) + LEN(Align3))
YES
NO
Type1
Type4: Wierd
Type2
Type3
Type1
Fragment
Read1
Read2
Type2
Fragment
Read1
Read2
Type3
Fragment
Read1
Read2

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 sepraruted 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: 100000
  -l LENGTH, --length LENGTH
                        shortest length to be considered for each part of the
                        pair, default: 15

Library dependency: Bio, itertools
```

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.

2.2.6 Step 6: Determine strong interactions.

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.

CHAPTER

THREE

INDICES AND TABLES

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