CRSSANT: Cross-linked RNA Secondary Structure Analysis using Network Techniques

1. Overview

RNA crosslinking, proximity ligation and high throughput sequencing produces non-continuous reads that indicate base pairing and higher order interactions, either in RNA secondary structures or intermolecular complexes. CRSSANT (pronounced 'croissant') is a computational pipeline for analyzing non-continuous/gapped reads from a variety of methods that employ the crosslink-ligation principle, including PARIS, LIGR, SPLASH, COMRADES, hiCLIP, etc. CRSSANT optimizes short-read mapping, automates alignment processing, and clusters gap1 and trans alignments into duplex groups (DG) and non-overlapping groups (NG). More complex arrangments are assembled into higher level structures. In particular gapm alignments with 2 gaps or 3 segments are assembled into tri-segment groups (TGs). Overlapping alignments are used to discover homotypic interactions (RNA homodimers).

Briefly, the CRSSANT pipeline operates as follows. First, sequencing reads that have been processed to remove adapters are mapped references with STAR and a new set of optimized options. Second, alignments are filtered, rearranged and classified into different types (gaptypes.py and gapfilter.py). Third, we use network analysis methods to cluster non-continuous alignments into DGs and calculate the confidence for each DG. The DGs are used as the foundation for the assembly of TGs.

CRSSANT is written in Python and available as source code that you can download and run directly on your own machine (no compiling needed). An earlier version of the DG assembly method is available here: (https://github.com/ihwang/CRSSANT). For more about the CRSSANT pipeline, please see the bioRxiv preprint by Zhang et al. 2021.

The CRSSANT pipeline consists of five steps:

- Step 1: Classify alignments
- Step 2: Segment and gap statistics
- Step 3: Filter spliced and short gaps
- Step 4: Cluster gap1 and trans alignments to DGs
- Step 5: Cluster gapm alignments to TGs

2. System and environment requirements

2.1 Download and prepare environment

Download the scripts and save it to a known path/location. No special installation is needed, but the python package dependencies need to be properly resolved before use. You will need Python version 3.6+ and the following Python packages. We recommend downloading the latest versions of these packages using the Ananconda/Bioconda package manager. Currently, the NetworkX version only works with python 3.6, but not higher versions.

- NetworkX v2.1+ (Anaconda link)
- NumPy (Anaconda link)
- SciPy (Anaconda link)
- scikit-learn (Anaconda link)

Additional tools for used for mapping and general processing of high throughput sequencing data, including STAR, samtools and bedtools.

- STAR v2.7.1+
- samtools v1.1+
- bedtools v2.22+

For visualization of the results, we recommend IGV, which has features for grouping alignments based on tags, such as DG and NG that we implemented here. IGV can also directly visualize DG summary information and RNA secondary structures. VARNA is recommended for visualizing RNA secondary structures in a variety of formats, including

- IGV v2.4+
- VARNA v1.0+

2.2 System requirements and tests

The programs are generally run in x86-64 compatible processors, including 64 bit Linux or Mac OS X, if there is enough memory. Read mapping against mammalian genomes using STAR requires at least 30G memory. Alignment classification typically requires 100G memory. As a result, these two steps should be run in a cluster with large memory.

Test datasets and example output files are provided for all steps except STAR mapping, which is a well maintained package. Test files are located in the example folder. The analysis pipeline is preferably run as separate steps to allow maximal control and quality assurance. In addition, shell scripts for a typical pipeline are also provided as examples.

3. Running CRSSANT on a local mechine

Before running CRSSANT, the input files (such as the ones provided in the demo dataset) need to be placed in a single location. Users can process any crosslink-ligation data using the CRSSANT pipeline. Navigate to a directory of your choice in your local machine and place the following files in the directory:

- **BAM/SAM** file containing aligned reads from the sample of interest. Ensure that the BAM/SAM file was mapped using STAR with a set optimized parameters as follows:

  STAR --runMode alignReads --genomeDir /path/to/index --readFilesIn /path/to/reads/files --outFileNamePrefix /path/to/output/prefix --genomeLoad NoSharedMemory --outReadsUnmapped Fastx --outFilterMultimapNmax 10 --outFilterScoreMinOverLread 0 --outSAMattributes All --outSAMtype BAM Unsorted SortedByCoordinate --alignIntronMin 1 --scoreGap 0 --scoreGapNoncan 0 --scoreGapGCAG 0 --scoreGapATAC 0 --scoreGenomicLengthLog2scale -1 --chimOutType WithinBAM HardClip --chimSegmentMin 5 --chimJunctionOverhangMin 5 --chimScoreJunctionNonGTAG 0 -- chimScoreDropMax 80 --chimNonchimScoreDropMin 20

- **Gene bed** file should contain six required BED fileds: Chrom, ChromStart, ChromEnd, Name, Score, Strand.

- **GTF** file, the annotation file containing the splicing junctions should be in GTF format.

Now that all the input files are ready, the CRSSANT pipeline can be run by using the following command, after navigating to the folder that contains the Python scripts:

$ python run_CRSSANT.py input_dir output_dir sample_name Gtf idloc genesfile outprefix

- InputDir: Full path to the directory where the input files exist
- OutputDir: Full path to the directory where the output files will be saved
- SampleName: Name of the sample
- Gtf: The annotation file containing the splicing junctions
- Idloc: Column number of the transcript_ID field in GTF files, is usually field 11.
- genesfile: gene annotations (BED)
- outprefix: output prefix

4. Output of CRSSANT

The output files from CRSSANT are gathered in four folders:

- *alignments_classsify*: created at step of classify alignments
- *alignments_statistics*: created at step of Segment and gap statistics
- *alignments_DGs*: created at step of DGs assembly
- *alignments_TGs*: created at step of TGs assembly

4.1 alignments_classsify

Successful completion of this step results in 7 files. All of these sam files can be converted to sorted bam for visualization on IGV.

- cont.sam: continuous alignments
- gap1.sam: non-continuous alignments, each has 1 gap
- gapm.sam: non-continuous alignments, each has more than 1 gaps
- trans.sam: non-continuous alignments with the 2 arms on different strands or chromosomes
- homo.sam: non-continuous alignments with the 2 arms overlapping each other
- bad.sam: non-continuous alignments with complex combinations of indels and gaps
- log.out: log file for the run, including input and output alignment counts (for gap1, gapm, trans, homo and bad)

4.2 alignments_statistics

- GapLen_distribution.pdf: Gap length distribution
- SegLen_distribution.pdf: Segment length distribution

4.3 alignments_DGs

After DG clustering, crssant.py verifies that the DGs do not contain any non-overlapping reads, i.e. any reads where the start position of its left arm is greater than or equal to the stop position of the right arm of any other read in the DG. If the DGs do not contain any non-overlapping reads, then the following output files ending in the following are written:

- .sam: SAM file containing alignments that were successfully assigned to DGs, plus DG and NG annotations
- dg.bedpe: bedpe file listing all duplex groups.

The sam output can be converted to bam for visualization in IGV, where DG and NG tags can be used to sort and group alignments. The bed output file can be visualized in IGV, where the two arms of each DG can be visualized as two 'exons', or as an arc the connects far ends of the DG (http://software.broadinstitute.org/software/igv/node/284).

DGs with the 2 arms on the same strand and chromosome can be further converted to bed12 for visualization on IGV. The fields of the BED12 file are used according to the standard definition with the exception of fields 4 and 5. Field 5 score (or field 8 in bedpe) is defined as the number of non-continuous alignments in this DG. Field 4 name (or field 7 in bedpe) is defined in the format gene1,gene2_DGID_covfrac, where gene1,gene2 represents the two genes that this DG connects, DGID is a numerical ID of the DG, covfrac is the confidence of the DG, defined as c / sqrt(a*b) and

c = number of reads in a given DG
a = number of reads overlapping the left arm of the DG
b = number of reads overlapping the right arm of the DG

4.4 alignments_TGs

TG clustering produces a sam file just like the gapm input sam file, except the addition of a new TG tag at the end of each alignment record. The output sam files can be converted to sorted bam for visualization on IGV. The TG tag can be used to group and sort alignments.