nature portfolio

https://doi.org/10.1038/s41596-021-00651-w

Supplementary information

Capture-C: a modular and flexible approach for high-resolution chromosome conformation capture

In the format provided by the authors and unedited

nature portfolio

https://doi.org/10.1038/s41596-021-00651-w

Supplementary information

Capture-C: a modular and flexible approach for high-resolution chromosome conformation capture

In the format provided by the authors and unedited

ccanalyser

Release 0.0.1

asmith, dsims

Apr 15, 2021

CONTENTS

CCanalyser is a package explicitly designed for processing Capture-C, Tri-C and Tiled-C data. It consists of a configurable data processing pipeline and a supporting command line interface to enable fine-grained control over the analysis. Unlike other pipelines that are designed to process Hi-C or Capture-HiC data, the filtering steps in ccanalyser are specifically optimised Capture-C, Tri-C and Tiled-C. This package is effectively a python based re-implementation of CCseqBasicS, Tri-C, Tiled-C and CaptureCompare designed to provide an end-to-end solution for processing data from these three assays.

CHAPTER

ONE

INSTALLING

1.1 Pre-Installation recommendations

1. Install conda if it has not been already using the conda install instructions.
2. If ccanalyser is **not** being installed through conda, first generate a new conda environment using the yaml file in the GitHub repository:

```
conda env create -f ccanalyser_conda_env.yml
conda activate cc
```

3. If you intend to use a cluster e.g. SunGrid engine/SLURM add the path to the DRMAA interface to your .bashrc:

```
# Access to the DRMAA library: https://en.wikipedia.org/wiki/DRMAA
echo "export DRMAA_LIBRARY_PATH=/<full-path>/libdrmaa.so" >> ~/.bashrc

# You can get this value from your configured environment:
env | grep DRMAA_LIBRARY_PATH

# or just look for the library:
find / -name "*libdrmaa.so"
```

1.2 Installation

The package can be installed in several ways:

Note: Currently only github installation is supported

1. Install from conda:

```
conda install ccanalyser # do not use: not available yet
```

2. Install from pypi:

```
pip install ccanalyser # do not use: not available yet
```

3. Install from GitHub:

```
git clone https://github.com/sims-lab/capture-c.git
cd capture-c
pip install .
```

CHAPTER

TWO

PIPELINE

The main feature of ccanalyser is the end-to-end data processing pipeline. The pipeline has been written using the cgat-core workflow management system and the following diagram illustrates the steps performed by the pipeline:

This section provides further details on how to run the pipeline. In essence the pipeline requires a working directory with correctly named FASTQ files and a *config.yml* file that provides the pipeline configuration.

2.1 Step 1 - Create a working directory

To run the pipeline you will need to create a working directory for the pipeline run:

```
mkdir RS411_EPZ5676/
cd RS411_EPZ5676/
```

The pipeline will be executed here and all files will be generated in this directory.

2.2 Step 2 - Edit a copy of config.yml

The configuration file config.yml enables parameterisation of the pipeline run with user specific settings. Furthermore, it also provides paths to essential files for the pipeline run (e.g., bowtie2 indices). The paths supplied do not have to be in the same directory as the pipeline but it is recomended to copy the capture viewpoints used to the working directory.

> **Warning:** The yaml file must be named **config.yml** for the pipeline to recognise it and run correctly.

This yaml file can be edited using standard text editors e.g.:

```
# To use gedit
gedit config.yml

# To use nano
nano config.yml
```

2.3 Step 3 - Copy or link fastq files into the working directory

The pipeline requires that fastq files are paired and in any of these formats:

Note: Gziped files are handled appropriately without the need for extraction if .gz is present at the end of the file name.

Note: Multi-lane FASTQ files should be concatenated prior to running the pipeline; otherwise multiple separate analyses will be performed.

Here is an example of file pairing for two samples:

- samplename1_R1.fastq.gz
- samplename1_R2.fastq.gz
- samplename2_1.fastq
- samplename2_2.fastq

All FASTQ files present in the directory will be processed by the pipeline in parallel and original FASTQ files will not be modified. If new FASTQ files are added to a pre-run pipeline, only the new files will be processed.

Copy:

```
cp PATH_TO_FASTQ/example_R1.fastq.gz.
```

Symlink example:

Be sure to use the absolute path for symlinks

```
ln -s /ABSOLUTE_PATH_TO_FASTQ/example_R1.fastq.gz
```

2.4 Step 4 - Running the pipeline

After copying/linking FASTQ files into the working directory and configuring the copy of config.yml in the working directory for the current experiment, the pipeline can be run with:

```
ccanalyser pipeline
```

There are several options to visualise which tasks will be performed by the pipeline before running.

The tasks to be performed can be examined with:

```
# Shows the tasks to be performed
ccanalyser pipeline show

# Plots a directed graph using graphviz
ccanalyser pipeline plot
```

If you are happy with the tasks to be performed, the full pipeline run can be launched with:

```
# If using all default settings and using a cluster
ccanalyser pipeline make

# If not using a cluster, run in local mode.
ccanalyser pipeline make --local -p 4

# Avoiding network disconnections
nohup ccanalyser pipeline make &
```

See cgat-core Read the Docs for additional information.

2.5 Step 5 - Running the pipeline to a specified stage

There are currently multiple stopping points built into the pipeline at key stages. These are:

- `fastq_preprocessing` - Stops after *in silico* digestion of FASTQ files.
- `pre_annotation` - Stops before aligned slices are ready to be annotated.
- `post_annotation` - Stops after aligned slices have been annotated.
- `post_ccanalyser_analysis` - Stops after reporters have been identified and duplicate filtered.
- `full` - Run the pipeline until all required tasks are complete.

To run the pipeline until one of these stopping points, use:

```
# Run until TASK_NAME step
ccanalyser pipeline make TASK_NAME
```

CHAPTER

THREE

PIPELINE FAQ

3.1 Interruptions to the pipeline

If for any reason the pipeline interrupted, simply delete any malformed file or directory and re-run the pipeline. The pipeline will then continue with the analysis without needing to start from the beginning.

Leftover ctmpXXX.sh files can be safely deleted if the pipeline has stopped running:

```
rm -f *.sh*
```

To restart the pipeline from the beginning for any reason, simply delete all processed files and re-run the pipeline:

```
rm -rf ccanalyser_preprocessing/ ccanalyser_analysis/ ccanalyser_compare/ statistics/␣
↪pipeline.log
ccanalyser pipeline make
```

3.2 DRMAA error

A common error when running the pipeline is:

```
GLOBAL_SESSION = drmaa.Session()
NameError: name 'drmaa' is not defined
```

This error occurrs because you are not connected to the cluster. See *Pre-Installation recommendations* for how to add DRMAA_LIBRARY_PATH to your bashrc. Alternatively, if you do not want to use a cluster you can run the pipeline in *local mode*.

3.3 KeyError: 'missing parameter accessed'

This occurs if one of the keys in config.yml has been altered or deleted. Either find the source of the error or get a new version of the yaml file and fill it out.

3.4 ValueError: not enough values to unpack (expected 2, got 1)

The fastq files are not paired correctly. Please ensure that fastq files are in the format:

```
NAME-OF-SAMPLE_(R)1.fastq.gz
NAME-OF-SAMPLE_(R)2.fastq.gz
```

CHAPTER

FOUR

CLI DOCUMENTATION

4.1 ccanalyser

An end to end solution for processing: Capture-C, Tri-C and Tiled-C data.

```
ccanalyser [OPTIONS] COMMAND [ARGS]...
```

4.1.1 alignments

Alignment annotation, identification and deduplication.

```
ccanalyser alignments [OPTIONS] COMMAND [ARGS]...
```

annotate

Annotates a bed file with other bed files using bedtools intersect.

Whilst bedtools intersect allows for interval names and counts to be used for annotating intervals, this command provides the ability to annotate intervals with both interval names and counts at the same time. As the pipeline allows for empty bed files, this command has built in support to deal with blank/malformed bed files and will return default N/A values.

Prior to interval annotation, the bed file to be intersected is validated and duplicate entries/multimapping reads are removed to ensure consistent annotations and prevent issues with reporter identification.

```
ccanalyser alignments annotate [OPTIONS] SLICES
```

Options

-a, --actions <actions>
Determines if the overlaps are counted or if the name should just be reported

Options get | count

-b, --bed_files <bed_files>
Bed file(s) to intersect with slices

-n, --names <names>
Names to use as column names for the output tsv file.

-f, --overlap_fractions <overlap_fractions>
The minimum overlap required for an intersection between two intervals to be reported.

-o, --output <output>
Path for the annotated slices to be output.

--duplicates <duplicates>
Method to use for reconciling duplicate slices (i.e. multimapping). Currently only 'remove' is supported.

Options remove

-p, --n_cores <n_cores>
Intersections are performed by chromosome, this determines the number of cores.

--invalid_bed_action <invalid_bed_action>
Method to deal with invalid bed files e.g. blank or incorrectly formatted. Setting this to 'ignore' will report default N/A values (either '.' or 0) for invalid files

Options ignore | error

Arguments

SLICES
Required argument

deduplicate

Identifies and removes duplicated aligned fragments.

PCR duplicates are very commonly present in Capture-C/Tri-C/Tiled-C data and must be removed for accurate analysis. Unlike fastq deduplicate, this command removes fragments with identical genomic coordinates.

Non-combined (pe) and combined (flashed) reads are treated slightly differently due to the increased confidence that the ligation junction has been captured for the flashed reads.

```
ccanalyser alignments deduplicate [OPTIONS] COMMAND [ARGS]...
```

identify

```
ccanalyser alignments deduplicate identify [OPTIONS] FRAGMENTS_FN
```

Options

-o, --output <output>
Path for outputting fragments with duplicated coordinates in json format.

--buffer <buffer>
Number of fragments to process at one time in order to preserve memory.

--read_type <read_type>
Indicates if the fragments have been combined (flashed) or not (pe).

Options flashed | pe

Arguments

FRAGMENTS_FN
Required argument

remove

Removes duplicated aligned fragments.

Parses a tsv file containing aligned read slices and outputs only slices from unique fragments. Duplicated parental read id determined by the "identify" subcommand are located within the slices tsv file and removed.

Outputs statistics for the number of unique slices and the number of duplicate slices identified.

```
ccanalyser alignments deduplicate remove [OPTIONS] SLICES_FN
```

Options

-d, --duplicated_ids <duplicated_ids>
Path to duplicated fragment ids determined by the 'identify' subcommand.

-o, --output <output>
Path for outputting deduplicated slices in tsv format.

--buffer <buffer>
Number of fragments to process at one time, in order to preserve memory.

--stats_prefix <stats_prefix>
Output prefix for deduplication statistics

--sample_name <sample_name>
Name of sample being analysed e.g. DOX_treated_1. Required for correct statistics.

--read_type <read_type>
Indicates if the fragments have been combined (flashed) or not (pe). Required for correct statistics.

Options flashed | pe

Arguments

SLICES_FN
Required argument

filter

Removes unwanted aligned slices and identifies reporters.

Parses a BAM file and merges this with a supplied annotation to identify unwanted slices. Filtering can be tuned for Capture-C, Tri-C and Tiled-C data to ensure optimal filtering.

```
ccanalyser alignments filter [OPTIONS] [capture|tri|tiled]
```

Options

-b, --bam <bam>
Required Bam file to process

-a, --annotations <annotations>
Required Annotations for the bam file that must contain the required columns, see description.

--custom_filtering <custom_filtering>
Custom filtering to be used. This must be supplied as a path to a yaml file.

-o, --output_prefix <output_prefix>
Output prefix for deduplicated fastq file(s)

--stats_prefix <stats_prefix>
Output prefix for stats file(s)

--sample_name <sample_name>
Name of sample e.g. DOX_treated_1

--read_type <read_type>
Type of read

Options flashed | pe

--gzip, --no-gzip
Determines if files are gziped or not

Arguments

METHOD
Required argument

4.1.2 fastq

Fastq splitting, deduplication and digestion.

```
ccanalyser fastq [OPTIONS] COMMAND [ARGS]...
```

deduplicate

Identifies PCR duplicate fragments from Fastq files.

PCR duplicates are very commonly present in Capture-C/Tri-C/Tiled-C data and must be removed for accurate analysis. These commands attempt to identify and remove duplicate reads/fragments from fastq file(s) to speed up downstream analysis.

```
ccanalyser fastq deduplicate [OPTIONS] COMMAND [ARGS]...
```

identify

Identifies fragments with duplicated sequences.

Merges the hashed dictionaries (in json format) generated by the "parse" subcommand and identifies read with exactly the same sequence (share an identical hash). Duplicated read identifiers (hashed) are output in json format. The "remove" subcommand uses this dictionary to remove duplicates from fastq files.

```
ccanalyser fastq deduplicate identify [OPTIONS] [INPUT_FILES]...
```

Options

-o, --output <output>
Required Output file

Arguments

INPUT_FILES
Optional argument(s)

parse

Parses fastq file(s) into easy to deduplicate format.

This command parses one or more fastq files and generates a dictionary containing hashed read identifiers together with hashed concatenated sequences. The hash dictionary is output in json format and the identify subcommand can be used to determine which read identifiers have duplicate sequences.

```
ccanalyser fastq deduplicate parse [OPTIONS] INPUT_FILES...
```

Options

-o, --output <output>
File to store hashed sequence identifiers

--read_buffer <read_buffer>
Number of reads to process before writing to file

Arguments

INPUT_FILES
Required argument(s)

remove

Removes fragments with duplicated sequences from fastq files.

Parses input fastq files and removes any duplicates from the fastq file(s) that are present in the json file supplied. This json dictionary should be produced by the "identify" subcommand.

Statistics for the number of duplicated and unique reads are also provided.

```
ccanalyser fastq deduplicate remove [OPTIONS] [INPUT_FILES]...
```

Options

-o, --output_prefix <output_prefix>
Output prefix for deduplicated fastq file(s)

-d, --duplicated_ids <duplicated_ids>
Path to duplicate ids, identified by the identify subcommand

--read_buffer <read_buffer>
Number of reads to process before writing to file

--gzip, --no-gzip
Determines if files are gziped or not

--compression_level <compression_level>
Level of compression for output files

--sample_name <sample_name>
Name of sample e.g. DOX_treated_1

--stats_prefix <stats_prefix>
Output prefix for stats file

Arguments

INPUT_FILES
Optional argument(s)

digest

Performs in silico digestion of one or a pair of fastq files.

```
ccanalyser fastq digest [OPTIONS] INPUT_FASTQ...
```

Options

-r, --restriction_enzyme <restriction_enzyme>
Required Restriction enzyme name or sequence to use for in silico digestion.

-m, --mode <mode>
Required Digestion mode. Combined (Flashed) or non-combined (PE) read pairs.

Options flashed | pe

-o, --output_file <output_file>

-p, --n_cores <n_cores>

--minimum_slice_length <minimum_slice_length>

--keep_cutsite <keep_cutsite>

--compression_level <compression_level>
Level of compression for output files (1=low, 9=high)

--read_buffer <read_buffer>
Number of reads to process before writing to file to conserve memory.

--stats_prefix <stats_prefix>
Output prefix for stats file

--sample_name <sample_name>
Name of sample e.g. DOX_treated_1. Required for correct statistics.

Arguments

INPUT_FASTQ
Required argument(s)

split

Splits fastq file(s) into equal chunks of n reads.

```
ccanalyser fastq split [OPTIONS] INPUT_FILES...
```

Options

-m, --method <method>
Method to use for splitting

Options python | unix

-o, --output_prefix <output_prefix>
Output prefix for deduplicated fastq file(s)

--compression_level <compression_level>
Level of compression for output files

-n, --n_reads <n_reads>
Number of reads per fastq file

--gzip, --no-gzip
Determines if files are gziped or not

Arguments

INPUT_FILES
Required argument(s)

4.1.3 genome

Genome wide methods digestion.

```
ccanalyser genome [OPTIONS] COMMAND [ARGS]...
```

digest

Performs in silico digestion of a genome in fasta format.

Digests the supplied genome fasta file and generates a bed file containing the locations of all restriction fragments produced by the supplied restriction enzyme.

A log file recording the number of restriction fragments for the suplied genome is also generated.

```
ccanalyser genome digest [OPTIONS] INPUT_FASTA
```

Options

-r, --recognition_site <recognition_site>
Required Recognition enzyme or sequence

-l, --logfile <logfile>
Path for digestion log file

-o, --output_file <output_file>
Output file path

--remove_cutsite <remove_cutsite>
Exclude the recognition sequence from the output

--sort
Sorts the output bed file by chromosome and start coord.

Arguments

INPUT_FASTA
Required argument

4.1.4 pipeline

Runs the data processing pipeline

```
ccanalyser pipeline [OPTIONS] [make|show|clone|touch] [PIPELINE_OPTIONS]...
```

Options

-h, --help

Arguments

MODE
Required argument

PIPELINE_OPTIONS
Optional argument(s)

4.1.5 reporters

Reporter counting, storing, comparison, pileups and heatmaps.

```
ccanalyser reporters [OPTIONS] COMMAND [ARGS]...
```

count

Determines the number of captured restriction fragment interactions genome wide.

Parses a reporter slices tsv and counts the number of unique restriction fragment interaction combinations that occur within each fragment.

Options to ignore unwanted counts e.g. excluded regions or capture fragments are provided. In addition the number of reporter fragments can be subsampled if required.

```
ccanalyser reporters count [OPTIONS] REPORTERS
```

Options

-o, --output <output>
Name of output file

--remove_exclusions
Prevents analysis of fragments marked as proximity exclusions

--remove_capture
Prevents analysis of capture fragment interactions

--subsample <subsample>
Subsamples reporters before analysis of interactions

Arguments

REPORTERS
Required argument

differential

Identifies differential interactions between conditions.

Parses a union bedgraph containg reporter counts from at least two conditions with two or more replicates for a single capture probe and outputs differential interaction results. Following filtering to ensure that the number of interactions is above the required threshold (–threshold_count), diffxpy is used to run a wald test after fitting a negative binomial model to the interaction counts.The options to filter results can be filtered by a minimum mean value (threshold_mean) and/or maximum q-value (threshold-q) are also provided.

Notes:

> Currently both the capture viewpoints and the name of the probe being analysed must be provided in order to correctly extract cis interactions.
>
> If a N_SAMPLE * METADATA design matrix has not been supplied, the script assumes that the standard replicate naming structure has been followed i.e. SAMPLE_CONDITION_REPLICATE_(1|2).fastq.gz.

```
ccanalyser reporters differential [OPTIONS] UNION_BEDGRAPH
```

Options

-n, --capture_name <capture_name>
Required Name of capture probe, must be present in viewpoint file.

-c, --capture_viewpoints <capture_viewpoints>
Required Path to capture viewpoints bed file

-o, --output_prefix <output_prefix>
Output prefix for pairwise statistical comparisons

--design_matrix <design_matrix>
Path tsv file containing sample annotations (N_SAMPLES * N_INFO_COLUMNS)

--grouping_col <grouping_col>
Column to use for grouping replicates

--threshold_count <threshold_count>
Minimum count required to be considered for analysis

--threshold_q <threshold_q>
Upper threshold of q-value required for output.

--threshold_mean <threshold_mean>
Minimum mean count required for output.

Arguments

UNION_BEDGRAPH
Required argument

pileup

Extracts reporters from a capture experiment and generates a bedgraph file.

Identifies reporters for a single probe (if a probe name is supplied) or all capture probes present in a capture experiment HDF5 file.

The bedgraph generated can be normalised by the number of cis interactions for inter experiment comparisons and/or binned into even genomic windows.

```
ccanalyser reporters pileup [OPTIONS] COOLER_FN
```

Options

-n, --capture_names <capture_names>
Capture to extract and convert to bedgraph, if not provided will transform all.

-o, --output_prefix <output_prefix>
Output prefix for bedgraphs

--normalise
Normalised bedgraph (Correct for number of cis reads)

--binsize <binsize>
Binsize to use for converting bedgraph to evenly sized genomic bins

--gzip
Compress output using gzip

--scale_factor <scale_factor>
Scale factor to use for bedgraph normalisation

--sparse, --dense
Produce bedgraph containing just positive bins (sparse) or all bins (dense)

Arguments

COOLER_FN
Required argument

plot

Plots a heatmap of reporter interactions.

Parses a HDF5 file containg the result of a capture experiment (binned into even genomic windows) and plots a heatmap of interactions over a specified genomic range. If a capture probe name is not supplied the script will plot all probes present in the file.

Heatmaps can also be normalised (–normalise) using either:

- n_interactions: The number of cis interactions.

- **n_rf_n_interactions: Normalised to the number of restriction fragments making up both genomic bins** and by the number of cis interactions.
- **ice: ICE normalisation followed by number of cis interactions** correction.

```
ccanalyser reporters plot [OPTIONS] COOLER_FN
```

Options

-c, --coordinates <coordinates>
Coordinates in the format chr1:1000-2000 or a path to a .bed file with coordinates

-r, --resolution <resolution>
Required Resolution at which to plot. Must be present within the cooler file.

-n, --capture_names <capture_names>
Capture names to plot. If not supplied will plot all

--normalisation <normalisation>
Normalisation method for heatmap

Options n_interactions | n_rf_n_interactions | ice

--cmap <cmap>
Colour map to use for plotting

--vmax <vmax>
Vmax for plotting

--vmin <vmin>
Vmin for plotting

-o, --output_prefix <output_prefix>
Output prefix for plot

--remove_capture
Removes the capture probe bins from the matrix

Arguments

COOLER_FN
Required argument

store

Store reporter counts.

These commands store and manipulate reporter restriction fragment interaction counts as cooler formated groups in HDF5 files.

See subcommands for details.

```
ccanalyser reporters store [OPTIONS] COMMAND [ARGS]...
```

bins

Convert a cooler group containing restriction fragments to constant genomic windows

Parses a cooler group and aggregates restriction fragment interaction counts into genomic bins of a specified size. If the normalise option is selected, columns containing normalised counts are added to the pixels table of the output

```
ccanalyser reporters store bins [OPTIONS] COOLER_FN
```

Options

-b, --binsizes <binsizes>
Binsizes to use for windowing

--normalise
Enables normalisation of interaction counts during windowing

--overlap_fraction <overlap_fraction>
Minimum overlap between genomic bins and restriction fragments for overlap

-p, --n_cores <n_cores>
Number of cores used for binning

--scale_factor <scale_factor>
Scaling factor used for normalisation

--conversion_tables <conversion_tables>
Pickle file containing pre-computed fragment -> bin conversions.

-o, --output <output>
Name of output file. (Cooler formatted hdf5 file)

Arguments

COOLER_FN
Required argument

fragments

Stores restriction fragment interaction combinations at the restriction fragment level.

Parses reporter restriction fragment interaction counts produced by "ccanalyser reporters count" and gerates a cooler formatted group in an HDF5 File. See *https://cooler.readthedocs.io/en/latest/* for further details.

```
ccanalyser reporters store fragments [OPTIONS] COUNTS
```

Options

-f, --fragment_map <fragment_map>
Required Path to digested genome bed file

-c, --capture_viewpoints <capture_viewpoints>
Required Path to capture viewpoints file

-n, --capture_name <capture_name>
Required Name of capture viewpoint to store

-g, --genome <genome>
Name of genome

--suffix <suffix>
Suffix to append after the capture name for the output file

-o, --output <output>
Name of output file. (Cooler formatted hdf5 file)

Arguments

COUNTS
Required argument

merge

Merges ccanalyser cooler files together.

Produces a unified cooler with both restriction fragment and genomic bins whilst reducing the storage space required by hard linking the "bins" tables to prevent duplication.

```
ccanalyser reporters store merge [OPTIONS] COOLERS...
```

Options

-o, --output <output>
Output file name

Arguments

COOLERS
Required argument(s)

CHAPTER

FIVE

MODULE DOCUMENTATION

5.1 CCanalyser Modules

5.1.1 CCanalyser CLI Modules

alignments annotate

ccanalyser.cli.alignments_annotate.**cycle_argument**(*arg*)

Allows for the same argument to be stated once but repeated for all files

ccanalyser.cli.alignments_annotate.**remove_duplicates_from_bed**(*bed: Union[str, pybedtools.bedtool.BedTool, pandas.core.frame.DataFrame]*) → pybedtools.bedtool.BedTool

Simple removal of duplicated entries from bed file.

If a "score" field is present a higher scored entry is prioritised.

Parameters **bed** (*Union[str, BedTool, pd.DataFrame]*) – Bed object to deduplicate

Returns BedTool with deduplicated names

Return type BedTool

ccanalyser.cli.alignments_annotate.**annotate**(*slices: os.PathLike*, *actions: Optional[Tuple] = None*, *bed_files: Optional[Tuple] = None*, *names: Optional[Tuple] = None*, *overlap_fractions: Optional[Tuple] = None*, *output: Optional[os.PathLike] = None*, *duplicates: str = 'remove'*, *n_cores: int = 8*, *invalid_bed_action: str = 'error'*)

Annotates a bed file with other bed files using bedtools intersect.

Whilst bedtools intersect allows for interval names and counts to be used for annotating intervals, this command provides the ability to annotate intervals with both interval names and counts at the same time. As the pipeline allows for empty bed files, this command has built in support to deal with blank/malformed bed files and will return default N/A values.

Prior to interval annotation, the bed file to be intersected is validated and duplicate entries/multimapping reads are removed to ensure consistent annotations and prevent issues with reporter identification.

Parameters

- **slices** (*os.PathLike*) – Input bed file.
- **actions** (*Tuple, optional*) – Methods to use for annotation. Choose from (get|count). Defaults to None.
- **bed_files** (*Tuple, optional*) – Bed files to intersect with the bed file to be annotated. Defaults to None.
- **names** (*Tuple, optional*) – Column names for output tsv file. Defaults to None.
- **overlap_fractions** (*Tuple, optional*) – Minimum overlap fractions required to call an intersection. Defaults to None.
- **output** (*os.PathLike, optional*) – Output file path for annotated .tsv file. Defaults to None.
- **duplicates** (*str, optional*) – Method to deal with multimapping reads/duplicate bed names. Currently, "remove" is the only supported option. Defaults to "remove".
- **n_cores** (*int, optional*) – Number of corese to use for intersection. Bed files are split by chromosome for faster intersection. Defaults to 8.
- **invalid_bed_action** (*str, optional*) – Action to deal with invalid bed files. Choose from (ignore|error) .These can be ignored by setting to "ignore". Defaults to 'error'.

Raises **NotImplementedError** – Only supported option for duplicate bed names is remove.

alignments deduplicate

ccanalyser.cli.alignments_deduplicate.**identify**(*fragments_fn: os.PathLike*, *output: os.PathLike = 'duplicated_ids.json'*, *buffer: int = 1000000.0*, *read_type: str = 'flashed'*)

Identifies aligned fragments with duplicate coordinates.

Parses a tsv file containing filtered aligned fragments and generates a dictionary containing the hashed parental read id and hashed genomic coordinates of all slices. Duplicated fragments are implicitly removed if they share the same genomic coordinate hash.

For non-combined reads (pe) a genomic coordinate hash is generated from the start of the first slice and the end of the last slice. This is due to the decreased confidence in the quality of the centre of the fragment. The coordinate hash for combined reads (flashed) is generated directly from the fragment coordinates. Only fragments with the exact coordinates and slice order will be considered to be duplicates.

Identified duplicate fragments are output in json format to be used by the "remove" subcommand.

Parameters

- **fragments_fn** (*os.PathLike*) – Input fragments.tsv file to process.
- **output** (*os.PathLike, optional*) – Output path to output duplicated parental read ids. Defaults to "duplicated_ids.json".
- **buffer** (*int, optional*) – Number of fragments to process in memory. Defaults to 1e6.
- **read_type** (*str, optional*) – Process combined(flashed) or non-combined reads (pe). Due to the low confidence in the quaility of pe reads, duplicates are identified by removing any fragments with matching start and end coordinates. Defaults to "flashed".

ccanalyser.cli.alignments_deduplicate.**remove**(*slices_fn: os.PathLike*, *duplicated_ids: os.PathLike*, *output: os.PathLike = 'dedup.slices.tsv.gz'*, *buffer: int = 5000000.0*, *sample_name: str = ''*, *read_type: str = ''*, *stats_prefix: os.PathLike = ''*)

Removes duplicated aligned fragments.

Parses a tsv file containing aligned read slices and outputs only slices from unique fragments. Duplicated parental read id determined by the "identify" subcommand are located within the slices tsv file and removed.

Outputs statistics for the number of unique slices and the number of duplicate slices identified.

Parameters

- **slices_fn** (*os.PathLike*) – Input slices.tsv file.
- **duplicated_ids** (*os.PathLike*) – Duplicated parental read ids in json format.
- **output** (*os.PathLike, optional*) – Output file path for deduplicated slices. Defaults to "dedup.slices.tsv.gz".
- **buffer** (*int, optional*) – Number of slices to process in memory. Defaults to 1e6.
- **sample_name** (*str, optional*) – Name of sample being processed e.g. DOX-treated_1 used for statistics. Defaults to "".
- **read_type** (*str, optional*) – Process combined(flashed) or non-combined reads (pe) used for statistics. Defaults to "".
- **stats_prefix** (*os.PathLike, optional*) – Output path for deduplication statistics. Defaults to "".

alignments filter

ccanalyser.cli.alignments_filter.**merge_annotations**(*df: pandas.core.frame.DataFrame*, *annotations: os.PathLike*) → pandas.core.frame.DataFrame

Combines annotations with the parsed bam file output.

Uses pandas outer join on the indexes to merge annotations e.g. number of capture probe overlaps.

Annotation tsv must have the index as the first column and this index must have intersecting keys with the first dataframe's index.

Parameters

- **df** (*pd.DataFrame*) – Dataframe to merge with annotations
- **annotations** (*os.PathLike*) – Filename of .tsv to read and merge with df

Returns Merged dataframe

Return type pd.DataFrame

ccanalyser.cli.alignments_filter.**filter**(*bam: os.PathLike*, *annotations: os.PathLike*, *custom_filtering: os.PathLike = None*, *output_prefix: os.PathLike = 'reporters'*, *stats_prefix: os.PathLike = ''*, *method: str = 'capture'*, *sample_name: str = ''*, *read_type: str = ''*, *gzip: bool = False*)

Removes unwanted aligned slices and identifies reporters.

Parses a BAM file and merges this with a supplied annotation to identify unwanted slices. Filtering can be tuned for Capture-C, Tri-C and Tiled-C data to ensure optimal filtering.

Common filters include:

- Removal of unmapped slices
- Removal of excluded/blacklisted slices
- Removal of non-capture fragments
- Removal of multi-capture fragments
- Removal of non-reporter fragments
- Removal of fragments with duplicated coordinates.

For specific filtering for each of the three methods see:

- *CCSliceFilter*
- *TriCSliceFilter*
- *TiledCSliceFilter*

In addition to outputting valid reporter fragments and slices separated by capture probe, this script also provides statistics on the number of read/slices filtered at each stage, and the number of cis/trans reporters for each probe.

Notes

Whilst the script is capable of processing any annotations in tsv format, provided that the correct columns are present. It is highly recomended that the "annotate" subcomand is used to generate this file.

Slice filtering is currently hard coded into each filtering class. This will be modified in a future update to enable custom filtering orders.

Parameters

- **bam** (*os.PathLike*) – Input bam file to analyse
- **annotations** (*os.PathLike*) – Annotations file generated by slices-annotate
- **custom_filtering** (*os.PathLike*) – Allows for custom filtering to be performed. A yaml file is used to supply this ordering.
- **output_prefix** (*os.PathLike, optional*) – Output file prefix. Defaults to "reporters".
- **stats_prefix** (*os.PathLike, optional*) – Output stats prefix. Defaults to "".
- **method** (*str, optional*) – Analysis method. Choose from (captureltriltiled). Defaults to "capture".
- **sample_name** (*str, optional*) – Sample being processed e.g. DOX-treated_1. Defaults to "".
- **read_type** (*str, optional*) – Process combined(flashed) or non-combined reads (pe) used for statistics. Defaults to "".
- **gzip** (*bool, optional*) – Compress output with gzip. Defaults to False.

fastq deduplicate

Created on Fri Oct 4 13:47:20 2019 @author: asmith

ccanalyser.cli.fastq_deduplicate.**parse**(*input_files: Tuple*, *output: os.PathLike = 'out.json'*, *read_buffer: int = 100000.0*)

Parses fastq file(s) into easy to deduplicate format.

This command parses one or more fastq files and generates a dictionary containing hashed read identifiers together with hashed concatenated sequences. The hash dictionary is output in json format and the identify subcommand can be used to determine which read identifiers have duplicate sequences.

Parameters

- **input_files** (*Tuple*) – One or more fastq files to process
- **output** (*os.PathLike, optional*) – Output for parsed read identifiers and sequences. Defaults to "out.json".
- **read_buffer** (*int, optional*) – Number of reads to process before outputting to file. Defaults to 1e5.

ccanalyser.cli.fastq_deduplicate.**identify**(*input_files: Tuple*, *output: os.PathLike = 'duplicates.json'*)

Identifies fragments with duplicated sequences.

Merges the hashed dictionaries (in json format) generated by the "parse" subcommand and identifies read with exactly the same sequence (share an identical hash). Duplicated read identifiers (hashed) are output in json format. The "remove" subcommand uses this dictionary to remove duplicates from fastq files.

Parameters

- **input_files** (*Tuple*) – Paths to json files containing dictionaries with hashed read ids as the keys and hashed sequences as the values.
- **output** (*os.PathLike, optional*) – Duplicate read ids identified. Defaults to "duplicates.json".

ccanalyser.cli.fastq_deduplicate.**remove**(*input_files: Tuple*, *duplicated_ids: os.PathLike*, *read_buffer: int = 100000.0*, *output_prefix: os.PathLike = ''*, *gzip: bool = False*, *compression_level: int = 5*, *sample_name: str = ''*, *stats_prefix: os.PathLike = ''*)

Removes fragments with duplicated sequences from fastq files.

Parses input fastq files and removes any duplicates from the fastq file(s) that are present in the json file supplied. This json dictionary should be produced by the "identify" subcommand.

Statistics for the number of duplicated and unique reads are also provided.

Parameters

- **input_files** (*Tuple*) – Input fastq files (in the same order as used for the parse command).
- **duplicated_ids** (*os.PathLike*) – Duplicated read ids from identify command (hashed and in json format).
- **read_buffer** (*int, optional*) – Number of reads to process before writing to file. Defaults to 1e5.
- **output_prefix** (*os.PathLike, optional*) – Deduplicated fastq output prefix. Defaults to "".

- **gzip** (*bool, optional*) – Determines if output is gzip compressed using pigz. Defaults to False.
- **compression_level** (*int, optional*) – Level of compression if required (1-9). Defaults to 5.
- **sample_name** (*str, optional*) – Name of sample processed e.g. DOX-treated_1. Defaults to "".
- **stats_prefix** (*os.PathLike, optional*) – Output prefix for statistics. Defaults to "".

fastq digest

ccanalyser.cli.fastq_digest.**collate_statistics**(*statq: multiprocessing.context.BaseContext.Queue*, *n_subprocesses: int*) → pandas.core.frame.DataFrame

Collates digestion statistics from supplied statistics queue.

Parameters

- **statq** (*Queue*) – Queue to use for collating statistics. Final item(s) must be 'END'
- **n_subprocesses** (*int*) – Number of digestion processes used. Required to know when to stop aquiring data from the queue.

Returns

Digestion statistics in histogram format. Columns: 'read_type', 'read_number', 'unfiltered/filtered', 'n_slices', 'n_reads'

Return type pd.DataFrame

ccanalyser.cli.fastq_digest.**digest**(*input_fastq: Tuple*, *restriction_enzyme: str*, *mode: str = 'pe'*, *output_file: os.PathLike = 'out.fastq.gz'*, *minimum_slice_length: int = 18*, *compression_level: int = 5*, *n_cores: int = 1*, *read_buffer: int = 100000*, *stats_prefix: os.PathLike = ''*, *keep_cutsite: bool = False*, *sample_name: str = ''*)

Performs in silico digestion of one or a pair of fastq files.

Parameters

- **input_fastq** (*Tuple*) – Input fastq files to process
- **restriction_enzyme** (*str*) – Restriction enzyme name or site to use for digestion.
- **mode** (*str, optional*) – Digest combined(flashed) or non-combined(pe). Undigested pe reads are output but flashed are not written. Defaults to "pe".
- **output_file** (*os.PathLike, optional*) – Output fastq file path. Defaults to "out.fastq.gz".
- **minimum_slice_length** (*int, optional*) – Minimum allowed length for in silico digested reads. Defaults to 18.
- **compression_level** (*int, optional*) – Compression level for gzip output (1-9). Defaults to 5.
- **n_cores** (*int, optional*) – Number of digestion processes to use. Defaults to 1.

- **read_buffer** (*int, optional*) – Number of reads to process before writing to file. Defaults to 100000.
- **stats_prefix** (*os.PathLike, optional*) – Output prefix for stats file. Defaults to "".
- **keep_cutsite** (*bool, optional*) – Determines if cutsite is removed from the output. Defaults to False.
- **sample_name** (*str, optional*) – Name of sample processed eg. DOX-treated_1. Defaults to ''.

fastq split

Created on Wed Jan 8 15:45:09 2020

@author: asmith

Script splits a fastq into specified chunks

ccanalyser.cli.fastq_split.**run_unix_split**(*fn: os.PathLike*, *n_reads: int*, *read_number: int*, *output_prefix: os.PathLike = ''*, *gzip: bool = False*, *compression_level: int = 5*)

ccanalyser.cli.fastq_split.**split**(*input_files: Tuple*, *method: str = 'unix'*, *output_prefix: os.PathLike = 'split'*, *compression_level: int = 5*, *n_reads: int = 1000000*, *gzip: bool = True*)

Splits fastq file(s) into equal chunks of n reads.

Parameters

- **input_files** (*Tuple*) – Input fastq files to process.
- **method** (*str, optional*) – Python or unix method (faster but not guarenteed to mantain read pairings) to split the fastq files. Defaults to "unix".
- **output_prefix** (*os.PathLike, optional*) – Output prefix for split fastq files. Defaults to "split".
- **compression_level** (*int, optional*) – Compression level for gzipped output. Defaults to 5.
- **n_reads** (*int, optional*) – Number of reads to split the input fastq files into. Defaults to 1000000.
- **gzip** (*bool, optional*) – Gzip compress output files if True. Defaults to True.

genome digest

Created on Fri Oct 4 13:47:20 2019 @author: asmith

Script generates a bed file of restriction fragment locations in a given genome.

ccanalyser.cli.genome_digest.**parse_chromosomes**(*fasta: pysam.libcfaidx.FastxFile*) → Iterator[pysam.libcfaidx.FastqProxy]

Parses a whole genome fasta file and yields chromosome entries.

Parameters **fasta** (*pysam.FastxFile*) – Fasta file to process.

Yields *Iterator[pysam.FastqProxy]* – Chromosome entry.

ccanalyser.cli.genome_digest.**digest**(*input_fasta: os.PathLike*, *recognition_site: str*, *logfile: os.PathLike = 'genome_digest.log'*, *output_file: os.PathLike = 'genome_digest.bed'*, *remove_cutsite: bool = True*, *sort=False*)

Performs in silico digestion of a genome in fasta format.

Digests the supplied genome fasta file and generates a bed file containing the locations of all restriction fragments produced by the supplied restriction enzyme.

A log file recording the number of restriction fragments for the suplied genome is also generated.

Parameters

- **input_fasta** (*os.PathLike*) – Path to fasta file containing whole genome sequence, split by chromosome
- **recognition_site** (*str*) – Restriction enzyme name/ Sequence of recognition site.
- **logfile** (*os.PathLike, optional*) – Output path of the digestion logfile. Defaults to genome_digest.log.
- **output_file** (*os.PathLike, optional*) – Output path for digested chromosome bed file. Defaults to genome_digest.bed.
- **remove_cutsite** (*bool, optional*) – Determines if restriction site is removed. Defaults to True.

reporters count

ccanalyser.cli.reporters_count.**count_re_site_combinations**(*groups: pandas.core.groupby.groupby.GroupBy*, *column: str = 'restriction_fragment'*, *counts: Optional[collections.defaultdict] = None*) → collections.defaultdict

Counts the number of unique combinations bewteen groups in a column.

Parameters

- **df** (*pd.core.groupby.GroupBy*) – Aggregated dataframe for processing.
- **column** (*str, optional*) – Column to examine for unique combinations per group. Defaults to “restriction_fragment”.
- **counts** (*defaultdict, optional*) – defaultdict(int) containing previous counts. Defaults to None.

Returns defaultdict(int) containing the count of unique interactions.

Return type defaultdict

ccanalyser.cli.reporters_count.**count**(*reporters: os.PathLike*, *output: os.PathLike = 'counts.tsv'*, *remove_exclusions: bool = False*, *remove_capture: bool = False*, *subsample: int = 0*)

Determines the number of captured restriction fragment interactions genome wide.

Parses a reporter slices tsv and counts the number of unique restriction fragment interaction combinations that occur within each fragment.

Options to ignore unwanted counts e.g. excluded regions or capture fragments are provided. In addition the number of reporter fragments can be subsampled if required.

Parameters

- **reporters** (*os.PathLike*) – Reporter tsv file path.
- **output** (*os.PathLike, optional*) – Output file path for interaction counts tsv. Defaults to 'counts.tsv'.
- **remove_exclusions** (*bool, optional*) – Removes regions marked as capture proximity exclusions before counting. Defaults to False.
- **remove_capture** (*bool, optional*) – Removes all capture fragments before counting. Defaults to False.
- **subsample** (*int, optional*) – Subsamples the fragments by the specified fraction. Defaults to 0 i.e. No subsampling.

reporters differential

ccanalyser.cli.reporters_differential.**get_chromosome_from_name**(*df: pandas.core.frame.DataFrame*, *name: str*)

ccanalyser.cli.reporters_differential.**differential**(*union_bedgraph: os.PathLike*, *capture_name: str*, *capture_oligos: os.PathLike*, *output_prefix: os.PathLike = 'differential'*, *design_matrix: Optional[os.PathLike] = None*, *grouping_col: str = 'condition'*, *threshold_count: float = 20*, *threshold_q: float = 0.05*, *threshold_mean: float = 0*)

Identifies differential interactions between conditions.

Parses a union bedgraph containg reporter counts from at least two conditions with two or more replicates for a single capture probe and outputs differential interaction results. Following filtering to ensure that the number of interactions is above the required threshold (–threshold_count), diffxpy is used to run a wald test after fitting a negative binomial model to the interaction counts.The options to filter results can be filtered by a minimum mean value (threshold_mean) and/or maximum q-value (threshold-q) are also provided.

Notes

Currently both the capture oligos and the name of the probe being analysed must be provided in order to correctly extract cis interactions.

If a N_SAMPLE * METADATA design matrix has not been supplied, the script assumes that the standard replicate naming structure has been followed i.e. SAMPLE_CONDITION_REPLICATE_(1|2).fastq.gz.

Parameters

- **union_bedgraph** (*os.PathLike*) – Union bedgraph containg all samples to be compared.
- **capture_name** (*str*) – Name of capture probe. MUST match one probe within the supplied oligos.

- **capture_oligos** (*os.PathLike*) – Capture oligos used for the analysis.
- **output_prefix** (*os.PathLike, optional*) – Output prefix for differntial interactions. Defaults to 'differential'.
- **design_matrix** (*os.PathLike, optional*) – Design matrix to use for grouping samples. (N_SAMPLES * METADATA). Defaults to None.
- **grouping_col** (*str, optional*) – Column to use for grouping. Defaults to 'condition'.
- **threshold_count** (*float, optional*) – Minimum number of reported interactions required. Defaults to 20.
- **threshold_q** (*float, optional*) – Maximum q-value for output. Defaults to 0.05.
- **threshold_mean** (*float, optional*) – Minimum mean value for output. Defaults to 0.

reporters heatmap

ccanalyser.cli.reporters_heatmap.**plot_matrix**(*matrix*, *figsize=(10, 10)*, *axis_labels=None*, *cmap=None*, *vmin=0*, *vmax=0*)

ccanalyser.cli.reporters_heatmap.**plot**(*cooler_fn: os.PathLike*, *coordinates: Union[str, os.PathLike]*, *resolution: int*, *capture_names: Optional[Tuple] = None*, *normalisation: Optional[str] = None*, *cmap: str = 'jet'*, *vmax: float = 1*, *vmin: float = 0*, *output_prefix: os.PathLike = ''*, *remove_capture: bool = False*)

Plots a heatmap of reporter interactions.

Parses a HDF5 file containg the result of a capture experiment (binned into even genomic windows) and plots a heatmap of interactions over a specified genomic range. If a capture probe name is not supplied the script will plot all probes present in the file.

Heatmaps can also be normalised (–normalise) using either:

- raw: No normalisation is performed.
- n_interactions: The number of cis interactions.
- **n_rf_n_interactions: Normalised to the number of restriction fragments making up both genomic bins** and by the number of cis interactions.
- **ice: ICE normalisation followed by number of cis interactions** correction.

Parameters

- **cooler_fn** (*os.PathLike*) – Path to capture cooler file containing interactions
- **coordinates** (*Union[str, os.PathLike]*) – Coordinates for plotting. Either chrX:1000-2000 or bed file. If a bed file, the capture probe name must be contained in the name.
- **resolution** (*int*) – Genomic resolution to plot. Must be present in the cooler.
- **capture_names** (*Tuple, optional*) – Capture probes to plot. If None will plot all probes. Defaults to None.
- **normalisation** (*str, optional*) – Normalisation for heatmap. Choose from (n_interactions\n_rf_n_interactions\ice). Defaults to None.

- **cmap** (*str, optional*) – Colour map to use for heatmap. Defaults to "jet".
- **vmax** (*float, optional*) – vmaxold for heatmap. Defaults to 1.
- **output_prefix** (*os.PathLike, optional*) – Output prefix for heatmap. Defaults to "".

reporters pileup

ccanalyser.cli.reporters_pileup.**bedgraph**(*cooler_fn: os.PathLike*, *capture_names: Optional[list] = None*, *output_prefix: os.PathLike = ''*, *normalise: bool = False*, *binsize: int = 0*, *gzip: bool = True*, *scale_factor: int = 1000000.0*, *sparse: bool = True*)

Extracts reporters from a capture experiment and generates a bedgraph file.

Identifies reporters for a single probe (if a probe name is supplied) or all capture probes present in a capture experiment HDF5 file.

The bedgraph generated can be normalised by the number of cis interactions for inter experiment comparisons and/or binned into even genomic windows.

Parameters

- **cooler_fn** (*os.PathLike*) – Path to hdf5 file containing cooler groups.
- **capture_names** (*list, optional*) – Name of capture probe to extract. If None, will process all probes present in the file. Defaults to None.
- **output_prefix** (*os.PathLike, optional*) – Output file prefix for bedgraph. Defaults to "".
- **normalise** (*bool, optional*) – Normalise counts using the number of cis interactions. Defaults to False.
- **binsize** (*int, optional*) – Genomic binsize to use for generating bedgraph. No binning performed if less than 0. Defaults to 0.
- **gzip** (*bool, optional*) – Compress output bedgraph with gzip. Defaults to True.
- **scale_factor** (*int, optional*) – Scaling factor for normalisation. Defaults to 1e6.
- **sparse** (*bool, optional*) – Produce bedgraph containing just positive bins (True) or all bins (False). Defaults to True.

reporters store

ccanalyser.cli.reporters_store.**fragments**(*counts: os.PathLike*, *fragment_map: os.PathLike*, *output: os.PathLike*, *capture_name: str*, *capture_viewpoints: os.PathLike*, *genome: str = ''*, *suffix: str = ''*)

Stores restriction fragment interaction combinations at the restriction fragment level.

Parses reporter restriction fragment interaction counts produced by "ccanalyser reporters count" and gerates a cooler formatted group in an HDF5 File. See *https://cooler.readthedocs.io/en/latest/* for further details.

Parameters

- **counts** (*os.PathLike*) – Path to restriction fragment interactions counts .tsv file.

- **fragment_map** (*os.PathLike*) – Path to restriction fragment .bed file, generated with genome-digest command.
- **output** (*os.PathLike*) – Output file path for cooler hdf5 file.
- **capture_name** (*str*) – Name of capture probe.
- **capture_viewpoints** (*os.PathLike*) – Path to capture viewpoints bed file.
- **genome** (*str, optional*) – Name of genome used for alignment e.g. hg19. Defaults to "".
- **suffix** (*str, optional*) – Suffix to append to filename. Defaults to "".

ccanalyser.cli.reporters_store.**bins**(*cooler_fn: os.PathLike*, *output: os.PathLike*, *binsizes: Optional[Tuple] = None*, *normalise: bool = False*, *n_cores: int = 1*, *scale_factor: int = 1000000.0*, *overlap_fraction: float = 1e-09*, *conversion_tables: Optional[os.PathLike] = None*)

Convert a cooler group containing restriction fragments to constant genomic windows

Parses a cooler group and aggregates restriction fragment interaction counts into genomic bins of a specified size. If the normalise option is selected, columns containing normalised counts are added to the pixels table of the output

Notes

To avoid repeatedly calculating restriction fragment to bin conversions, bin conversion tables (a .pkl file containing a dictionary of *:class:ccanalyser.tools.storage.GenomicBinner* objects, one per binsize) can be supplied.

Parameters

- **cooler_fn** (*os.PathLike*) – Path to cooler file. Nested coolers can be specified by STORE_FN.hdf5::/PATH_TO_COOLER
- **output** (*os.PathLike*) – Path for output binned cooler file.
- **binsizes** (*Tuple, optional*) – Genomic window sizes to use for binning. Defaults to None.
- **normalise** (*bool, optional*) – Normalise the number of interactions to total number of cis interactions (True). Defaults to False.
- **n_cores** (*int, optional*) – Number of cores to use for binning. Performed in parallel by chromosome. Defaults to 1.
- **scale_factor** (*int, optional*) – Scaling factor to use for normalising interactions. Defaults to 1e6.
- **overlap_fraction** (*float, optional*) – Minimum fraction to use for defining overlapping bins. Defaults to 1e-9.

ccanalyser.cli.reporters_store.**merge**(*coolers: Tuple*, *output: os.PathLike*)

Merges ccanalyser cooler files together.

Produces a unified cooler with both restriction fragment and genomic bins whilst reducing the storage space required by hard linking the "bins" tables to prevent duplication.

Parameters

- **coolers** (*Tuple*) – Cooler files produced by either the fragments or bins subcommands.
- **output** (*os.PathLike*) – Path from merged cooler file.

tsv aggregate

Created on Wed Dec 11 21:49:19 2019

@author: davids, asmith

Join any number of tab delimited text files on a single column. Index column must have the same header name in each file to be joined. Performs an outer join on the index column. Assumes files have headers.

ccanalyser.cli.tsv_aggregate.**format_index_var**(*var*)

ccanalyser.cli.tsv_aggregate.**replace_na**(*df*)

ccanalyser.cli.tsv_aggregate.**is_compressed**(*files*)

ccanalyser.cli.tsv_aggregate.**load_tsv**(*tsv*, *index=None*, *header=None*)

ccanalyser.cli.tsv_aggregate.**join_tsvs**(*fnames*, *index_col*, *n_processes=8*, *header=True*)

ccanalyser.cli.tsv_aggregate.**concat_tsvs**(*fnames*, *delayed=False*, *header=None*)

ccanalyser.cli.tsv_aggregate.**main**(*input_files*, *output*, *index=None*, *header=None*, *method=None*, *n_processes=8*, *groupby_columns=None*, *aggregate_method=None*, *aggregate_columns=None*)

5.1.2 Pipeline

This pipeline processes data from Capture-C/NG Capture-C/Tri-C and Tiled-C sequencing protocols designed to identify 3D interactions in the genome from a specified viewpoint.

It takes Illumina paired-end sequencing reads in fastq format (gzip compression is prefered) as input and performs the following steps:

1. Identifies all restriction fragments in the genome
2. Quality control of raw reads (fastqc, multiqc)
3. Splits fastqs into smaller files to enable fast parallel processing.
4. Removal of PCR duplicates based on exact sequence matches from fastq files
5. Trimming of reads to remove adaptor sequence (trim_galore)
6. Combining overlapping read pairs (FLASh)
7. In silico digestion of reads in fastq files
8. Alignment of fastq files with a user specified aligner (i.e. bowtie/bowtie2; BWA is not supported)
9. Analysis of alignment statistics (picard CollectAlignmentSummaryMetrics, multiqc)
10. Annotation of mapped reads with overlaps of capture probes, exclusion regions, blacklist, restriction fragments
11. Removal of non-reporter slices and indentification of reporters
12. Removal of PCR duplicates (exact coordinate matches)
13. Storage of reporters in *cooler format <https://cooler.readthedocs.io/en/latest/datamodel.html>*
14. Generation of bedgraphs/BigWigs.
15. Collation of run statistics and generation of a run report

Optional:

- Generation of a UCSC track hub for visualisation.

- Differential interaction identification.
- Generation of subtraction bedgraphs for between condition comparisons
- Plotting of heatmaps.

@authors: asmith, dsims

ccanalyser.pipeline.pipeline.**modify_pipeline_params_dict**()
Modifies P.PARAMS dictionary.

- Selects the correct conda enviroment
- Ensures the correct queue manager is selected.
- Corrects the name of a UCSC hub by removing spaces and incorrect characters.

ccanalyser.pipeline.pipeline.**set_up_chromsizes**()
Ensures that genome chromsizes are present.

If chromsizes are not provided this function attempts to download them from UCSC. The P.PARAMS dictionary is updated with the location of the chromsizes.

ccanalyser.pipeline.pipeline.**genome_digest**(*infile*, *outfile*)
In silco digestion of the genome to identify restriction fragment coordinates.

Runs *ccanalyser genome digest*.

ccanalyser.pipeline.pipeline.**fastq_qc**(*infile*, *outfile*)
Runs fastqc on the input files to generate fastq statistics.

ccanalyser.pipeline.pipeline.**fastq_multiqc**(*infile*, *outfile*)
Collate fastqc reports into single report using multiqc

ccanalyser.pipeline.pipeline.**fastq_split**(*infiles*, *outfile*)
Splits the input fastq files into chunks for parallel processing

Runs *ccanalyser fastq split*.

ccanalyser.pipeline.pipeline.**fastq_duplicates_parse**(*infiles*, *outfile*, *sample_name*, *part_no*)
Parses fastq files into json format for sequence deduplication.

Runs *ccanalyser fastq deduplicate parse*

ccanalyser.pipeline.pipeline.**fastq_duplicates_identify**(*infiles*, *outfile*)
Identifies duplicate sequences from parsed fastq files in json format.

Runs *ccanalyser fastq deduplicate identify*

ccanalyser.pipeline.pipeline.**fastq_duplicates_remove**(*infiles*, *outfile*)
Removes duplicate read fragments identified from parsed fastq files.

ccanalyser.pipeline.pipeline.**stats_deduplication_collate**(*infiles*, *outfile*)
Combines deduplication statistics from fastq file partitions.

ccanalyser.pipeline.pipeline.**fastq_trim**(*infiles*, *outfile*)
Trim adaptor sequences from fastq files using trim_galore

ccanalyser.pipeline.pipeline.**stats_trim_collate**(*infiles*, *outfile*)
Extracts and collates adapter trimming statistics from trim_galore output

ccanalyser.pipeline.pipeline.**fastq_flash**(*infiles*, *outfile*)
Combine overlapping paired-end reads using FLASh

ccanalyser.pipeline.pipeline.**fastq_digest_combined**(*infile*, *outfile*)
In silico restriction enzyme digest of combined (flashed) read pairs

ccanalyser.pipeline.pipeline.**fastq_digest_non_combined**(*infiles*, *outfile*)
In silico restriction enzyme digest of non-combined (non-flashed) read pairs

ccanalyser.pipeline.pipeline.**stats_digestion_collate**(*infiles*, *outfile*)
Aggregates in silico digestion statistics from fastq file partitions.

ccanalyser.pipeline.pipeline.**fastq_preprocessing**()

ccanalyser.pipeline.pipeline.**fastq_alignment**(*infile*, *outfile*)
Aligns in silico digested fastq files to the genome.

ccanalyser.pipeline.pipeline.**alignments_merge**(*infiles*, *outfile*)
Combines bam files (by flashed/non-flashed status and sample).

This task simply provides an input for picard CollectAlignmentSummaryMetrics and is only used to provide overall mapping statistics. Fastq partitions are *not* combined at this stage.

ccanalyser.pipeline.pipeline.**alignments_index**(*infile*, *outfile*)
Indexes all bam files (both partitioned and merged)

ccanalyser.pipeline.pipeline.**pre_annotation**()

ccanalyser.pipeline.pipeline.**annotate_make_exclusion_bed**(*outfile*)
Generates exclusion window around each capture site

ccanalyser.pipeline.pipeline.**annotate_sort_viewpoints**(*outfile*)
Sorts the capture oligos for bedtools intersect with –sorted option

ccanalyser.pipeline.pipeline.**annotate_sort_blacklist**(*outfile*)
Sorts the capture oligos for bedtools intersect with –sorted option

ccanalyser.pipeline.pipeline.**annotate_alignments**(*infile*, *outfile*)
Annotates mapped read slices.

Slices are annotated with: * capture name * capture count * exclusion name * exclusion count * blacklist count * restriction fragment number

ccanalyser.pipeline.pipeline.**post_annotation**()
Runs the pipeline until just prior to identification of reporters

ccanalyser.pipeline.pipeline.**alignments_filter**(*infiles*, *outfile*)
Filteres slices and outputs reporter slices for each capture site

ccanalyser.pipeline.pipeline.**reporters_collate**(*infiles*, *outfile*, **grouping_args*)
Concatenates identified reporters

ccanalyser.pipeline.pipeline.**alignments_deduplicate_fragments**(*infile*, *outfile*, *read_type*)
Identifies duplicate fragments with the same coordinates and order.

ccanalyser.pipeline.pipeline.**alignments_deduplicate_slices**(*infile*, *outfile*, *sample_name*, *read_type*, *capture_oligo*)
Removes reporters with duplicate coordinates

ccanalyser.pipeline.pipeline.**alignments_deduplicate_collate**(*infiles*, *outfile*, **grouping_args*)
Final collation of reporters by sample and capture probe

ccanalyser.pipeline.pipeline.**stats_alignment_filtering_collate**(*infiles*, *outfile*)
'Combination of all reporter identification and filtering statistics

ccanalyser.pipeline.pipeline.**post_ccanalyser_analysis**()
Reporters have been identified, deduplicated and collated by sample/capture probe

ccanalyser.pipeline.pipeline.**reporters_count**(*infile*, *outfile*)
Counts the number of interactions identified between reporter restriction fragments

ccanalyser.pipeline.pipeline.**reporters_store_restriction_fragment**(*infile*, *outfile*, *sample_name*, *capture_name*)
Stores restriction fragment interaction counts in cooler format

ccanalyser.pipeline.pipeline.**generate_bin_conversion_tables**(*outfile*)
Converts restriction fragments to genomic bins.

Binning restriction fragments into genomic bins takes a substantial amount of time and memory. To avoid repeatedly performing the same action, bin conversion tables are calculated once for each required resolution and then stored as a pickle file.

ccanalyser.pipeline.pipeline.**reporters_store_binned**(*infile*, *outfile*, *capture_name*)
Converts a cooler file of restriction fragments to even genomic bins.

ccanalyser.pipeline.pipeline.**reporters_store_merged**(*infiles*, *outfile*, *sample_name*)
Combines cooler files together

ccanalyser.pipeline.pipeline.**pipeline_merge_stats**(*infiles*, *outfile*)
Generates a summary statistics file for the pipeline run.

ccanalyser.pipeline.pipeline.**pipeline_make_report**(*infile*, *outfile*)
Run jupyter notebook for reporting and plotting pipeline statistics

ccanalyser.pipeline.pipeline.**reporters_make_bedgraph**(*infile*, *outfiles*, *sample_name*)
Extract reporters in bedgraph format from stored interactions

ccanalyser.pipeline.pipeline.**reporters_make_bedgraph_normalised**(*infile*, *outfiles*, *sample_name*)
Extract reporters in bedgraph format from stored interactions.

In addition to generating a bedgraph this task also normalises the counts by the number of cis interactions identified to enable cross sample comparisons.

ccanalyser.pipeline.pipeline.**reporters_make_union_bedgraph**(*infiles*, *outfile*, *normalisation_type*, *capture_name*)
Collates bedgraphs by capture probe into a single file for comparison.

See bedtools unionbedg for more details.

ccanalyser.pipeline.pipeline.**reporters_make_subtraction_bedgraph**(*infile*, *outfile*)

ccanalyser.pipeline.pipeline.**reporters_make_bigwig**(*infile*, *outfile*)
Uses UCSC tools bedGraphToBigWig to generate bigWigs for each bedgraph

ccanalyser.pipeline.pipeline.**hub_make**(*infiles*, *outfile*, *statistics*)
Creates a ucsc hub from the pipeline output

ccanalyser.pipeline.pipeline.**hub_write_path**(*outfile*)
Convinence task to write hub url to use for adding custom hub to UCSC genome browser

ccanalyser.pipeline.pipeline.**identify_differential_interactions**(*infile*, *outfile*, *capture_name*)

ccanalyser.pipeline.pipeline.**reporters_plot_heatmap**(*infile*, *outfile*)
Plots a heatmap over a specified region

ccanalyser.pipeline.pipeline.**full**(*infiles*, *outfile*)

5.1.3 CCanalyser tools

ccanalyser.tools.annotate module

class ccanalyser.tools.annotate.**BedIntersection**(*bed1: Union[str, pybedtools.bedtool.BedTool, pandas.core.frame.DataFrame]*, *bed2: Union[str, pybedtools.bedtool.BedTool, pandas.core.frame.DataFrame]*, *intersection_name: str = 'count'*, *intersection_method: str = 'count'*, *intersection_min_frac: float = 1e-09*, *intersection_split_chrom: bool = True*, *n_cores: int = 1*, *invalid_bed_action='error'*)

Bases: object

Performs intersection between two named bed files.

Wrapper around bedtools intersect designed to intersect in parallel (by splitting file based on chromosome) and handle malformed bed files.

bed1
Bed file to intersect. Must be named.

Type Union[str, BedTool, pd.DataFrame]

bed2
Bed file to intersect.

Type Union[str, BedTool, pd.DataFrame]

intersection_name
Name for intersection.

Type str

min_frac
Minimum fraction required for intersection

Type float

n_cores
Number of cores for parallel intersection.

Type int

invalid_bed_action
Method to deal with missing/malformed bed files ("ignore"|"error")

property intersection
Intersects the two bed files and returns a pd.Series.

ccanalyser.tools.deduplicate module

class ccanalyser.tools.deduplicate.**ReadDeduplicationParserProcess**(*inq: multiprocessing.context.BaseContext.SimpleQueu outq: multiprocessing.context.BaseContext.SimpleQueu hash_seed: int = 42, save_hashed_dict_path: os.PathLike = 'parsed.json'*)

Bases: multiprocessing.context.Process

Process subclass for parsing fastq file(s) into a hashed {id:sequence} json format.

inq
Input read queue

outq
Output read queue (Not currently used)

hash_seed
Seed for xxhash64 algorithm to ensure consistency

save_hash_dict_path
Path to save hashed dictionary

run()
Processes fastq reads from multiple files and generates a hashed json dictionary.

Dictionary is hashed and in the format {(read 1 name + read 2 name): (sequence 1 + sequence 2)}.

Output path is specified by save_hashed_dict_path.

class ccanalyser.tools.deduplicate.**ReadDuplicateRemovalProcess**(*inq: multiprocessing.context.BaseContext.SimpleQueue, outq: multiprocessing.context.BaseContext.SimpleQueue, duplicated_ids: set, statq: Optional[multiprocessing.context.BaseCont = None, hash_seed: int = 42*)

Bases: multiprocessing.context.Process

Process subclass for parsing fastq file(s) and removing identified duplicates.

inq
Input read queue

outq
Output queue for deduplicated reads.

duplicated_ids
Concatenated read ids to remove from input fastq files.

statq
Output queue for statistics.

reads_total
Number of fastq reads processed.

reads_unique
Number of non-duplicated reads output.

hash_seed
Seed for xxhash algorithm. MUST be the same as used by ReadDuplicationParserProcess.

run()
Performs read deduplication based on sequence.

Unique reads are placed on outq and deduplication statistics are placed on statq.

ccanalyser.tools.digest module

class ccanalyser.tools.digest.**DigestedChrom**(*chrom: pysam.libcfaidx.FastqProxy*, *cutsite: str*, *fragment_number_offset: int = 0*, *fragment_min_len: int = 1*)

Bases: object

Performs in slico digestion of fasta files.

Identifies all restriction sites for a supplied restriction enzyme/restriction site and generates bed style entries.

chrom
Chromosome to digest

Type pysam.FastqProxy

recognition_seq
Sequence of restriction recognition site

Type str

recognition_len
Length of restriction recognition site

Type int

recognition_seq
Regular expression for restriction recognition site

Type re.Pattern

fragment_indexes
Indexes of fragment(s) start and end positions.

Type List[int]

fragment_number_offset
Starting fragment number.

Type int

fragment_min_len
Minimum fragment length required to report fragment

Type int

get_recognition_site_indexes() → List[int]
Gets the start position of all recognition sites present in the sequence.

Notes

Also appends the start and end of the sequnece to enable clearer itteration through the indexes.

Returns Indexes of fragment(s) start and end positions.

Return type List[int]

property fragments
Extracts the coordinates of restriction fragments from the sequence.

Yields *Iterator[Iterable[str]]* – Identified restriction fragments in bed format.

class ccanalyser.tools.digest.**DigestedRead**(*read: pysam.libcfaidx.FastqProxy*, *cutsite: str*, *min_slice_length: int = 18*, *slice_number_offset: int = 0*, *allow_undigested: bool = False*, *read_type: str = 'flashed'*)

Bases: object

Performs in slico digestion of fastq files.

Identifies all restriction sites for a supplied restriction enzyme/restriction site and generates bed style entries.

read
Read to digest.

Type pysam.FastqProxy

recognition_seq
Sequence of restriction recognition site.

Type str

recognition_len
Length of restriction recognition site.

Type int

recognition_seq
Regular expression for restriction recognition site.

Type re.Pattern

slices
List of Fastq formatted digested reads (slices).

Type List[str]

slice_indexes
Indexes of fragment(s) start and end positions.

Type List[int]

slice_number_offset
Starting fragment number.

Type int

min_slice_len
Minimum fragment length required to report fragment.

Type int

has_slices
Recognition site(s) present within sequence.

Type bool

get_recognition_site_indexes() → List[int]

class ccanalyser.tools.digest.**ReadDigestionProcess**(*inq: multiprocessing.context.BaseContext.SimpleQueue*, *outq: multiprocessing.context.BaseContext.SimpleQueue*, *statq: Optional[multiprocessing.context.BaseContext.Queue] = None*, ***digestion_kwargs*)

Bases: multiprocessing.context.Process

Process subclass for multiprocessing fastq digestion.

run()

Performs read digestion.

Reads to digest are pulled from inq, digested with the DigestedRead class and the results placed on outq for writing.

If a statq is provided, read digestion stats are placed into this queue for aggregation.

ccanalyser.tools.filter module

class ccanalyser.tools.filter.**SliceFilter**(*slices: pandas.core.frame.DataFrame*, *filter_stages: Optional[dict] = None*, *sample_name: str = ''*, *read_type: str = ''*)

Bases: object

Perform slice filtering (inplace) and reporter identification.

The SliceFilter classes e.g. CCSliceFilter, TriCSliceFilter, TiledCSliceFilter perform all of the filtering (inplace) and reporter identification whilst also providing statistics of the numbers of slices/reads removed at each stage.

slices

Annotated slices dataframe.

Type pd.DataFrame

fragments

Slices dataframe aggregated by parental read.

Type pd.DataFrame

reporters

Slices identified as reporters.

Type pd.DataFrame

filter_stages

Dictionary containg stages and a list of class methods (str) required to get to this stage.

Type dict

slice_stats

Provides slice level statistics.

Type pd.DataFrame

read_stats

Provides statistics of slice filtering at the parental read level.

Type pd.DataFrame

filter_stats
Provides statistics of read filtering.

Type pd.DataFrame

property filters
A list of the callable filters present within the slice filterer instance.

Returns All filters present in the class.

Return type list

property slice_stats
Statistics at the slice level.

Returns Statistics per slice.

Return type pd.DataFrame

property filter_stats
Statistics for each filter stage.

Returns Statistics of the number of slices removed at each stage.

Return type pd.DataFrame

property read_stats
Gets statistics at a read level.

Aggregates slices by parental read id and calculates stats.

Returns Statistics of the slices/fragments removed aggregated by read id.

Return type pd.DataFrame

property fragments
Summarises slices at the fragment level.

Uses pandas groupby to aggregate slices by their parental read name (shared by all slices from the same fragment). Also determines the number of reporter slices for each fragment.

Returns Slices aggregated by parental read name.

Return type pd.DataFrame

property reporters
Extracts reporter slices from slices dataframe i.e. non-capture slices

Returns All non-capture slices

Return type pd.DataFrame

filter_slices(*output_slices=False*, *output_location='.'*)
Performs slice filtering.

Filters are applied to the slices dataframe in the order specified by filter_stages. Filtering stats aggregated at the slice and fragment level are also printed.

Parameters

- **output_slices** (*bool, optional*) – Determines if slices are to be output to a specified location after each filtering step. Useful for debugging. Defaults to False.
- **output_location** (*str, optional*) – Location to output slices at each stage. Defaults to ".".

get_unfiltered_slices()
Does not modify slices.

remove_unmapped_slices()
Removes slices marked as unmapped (Uncommon)

remove_orphan_slices()
Remove fragments with only one aligned slice (Common)

remove_duplicate_re_frags()
Prevent the same restriction fragment being counted more than once (Uncommon).

Example

–RE_FRAG1–-—Capture—-–RE_FRAG1—-

remove_slices_without_re_frag_assigned()
Removes slices if restriction_fragment column is N/A

remove_duplicate_slices()
Remove all slices if the slice coordinates and slice order are shared.

This method is designed to remove a fragment if it is a PCR duplicate (Common).

Example

Frag 1: chr1:1000-1250 chr1:1500-1750
Frag 2: chr1:1000-1250 chr1:1500-1750
Frag 3: chr1:1050-1275 chr1:1600-1755
Frag 4: chr1:1500-1750 chr1:1000-1250

Frag 2 removed. Frag 1,3,4 retained

remove_duplicate_slices_pe()
Removes PCR duplicates from non-flashed (PE) fragments (Common).

Sequence quality is often lower at the 3' end of reads leading to variance in mapping coordinates. PCR duplicates are removed by checking that the fragment start and end are not duplicated in the dataframe.

remove_excluded_slices()
Removes any slices in the exclusion region (default 1kb) (V. Common)

remove_blacklisted_slices()
Removes slices marked as being within blacklisted regions

class ccanalyser.tools.filter.**CCSliceFilter**(*slices*, *filter_stages=None*, ***sample_kwargs*)
Bases: *ccanalyser.tools.filter.SliceFilter*

Perform Capture-C slice filtering (inplace) and reporter identification.

SliceFilter tuned specifically for Capture-C data. This class has addtional methods to remove common artifacts in Capture-C data i.e. multi-capture fragments, non-reporter fragments, multi-capture reporters. The default filter order is as follows:

- remove_unmapped_slices
- remove_orphan_slices

- remove_multi_capture_fragments
- remove_excluded_slices
- remove_blacklisted_slices
- remove_non_reporter_fragments
- remove_multicapture_reporters
- remove_slices_without_re_frag_assigned
- remove_duplicate_re_frags
- remove_duplicate_slices
- remove_duplicate_slices_pe
- remove_non_reporter_fragments

See the individual methods for further details.

slices
Annotated slices dataframe.

Type pd.DataFrame

fragments
Slices dataframe aggregated by parental read.

Type pd.DataFrame

reporters
Slices identified as reporters.

Type pd.DataFrame

filter_stages
Dictionary containg stages and a list of class methods (str) required to get to this stage.

Type dict

slice_stats
Provides slice level statistics.

Type pd.DataFrame

read_stats
Provides statistics of slice filtering at the parental read level.

Type pd.DataFrame

filter_stats
Provides statistics of read filtering.

Type pd.DataFrame

property fragments
Summarises slices at the fragment level.

Uses pandas groupby to aggregate slices by their parental read name (shared by all slices from the same fragment). Also determines the number of reporter slices for each fragment.

Returns Slices aggregated by parental read name.

Return type pd.DataFrame

property slice_stats
Statistics at the slice level.

Returns Statistics per slice.

Return type pd.DataFrame

property frag_stats
Statistics aggregated at the fragment level.

As this involves slice aggregation it can be rather slow for large datasets. It is recomended to only use this property if it is required.

Returns Fragment level statistics

Return type pd.DataFrame

property reporters
Extracts reporter slices from slices dataframe i.e. non-capture slices

Returns All non-capture slices

Return type pd.DataFrame

property captures
Extracts capture slices from slices dataframe

i.e. slices that do not have a null capture name

Returns Capture slices

Return type pd.DataFrame

property capture_site_stats
Extracts the number of unique capture sites.

property merged_captures_and_reporters
Merges captures and reporters sharing the same parental id.

Capture slices and reporter slices with the same parental read id are merged together. The prefixes 'capture' and 'reporter' are used to identify slices marked as either captures or reporters.

Returns Merged capture and reporter slices

Return type pd.DataFrame

property cis_or_trans_stats
Extracts reporter cis/trans statistics from slices.

Returns Reporter cis/trans statistics

Return type pd.DataFrame

remove_non_reporter_fragments()
Removes the fragment if it has no reporter slices present (Common)

remove_multi_capture_fragments()
Removes double capture fragments.

All slices (i.e. the entire fragment) are removed if more than one capture probe is present i.e. a double capture (V. Common)

remove_multicapture_reporters(*n_adjacent: int = 1*)
Deals with an odd situation in which a reporter spanning two adjacent capture sites is not removed.

Example

——Capture 1—-/——Capture 2—— ——REP———

In this case the "reporter" slice is not considered either a capture or exclusion.

These cases are dealt with by explicitly removing reporters on restriction fragments adjacent to capture sites.

Parameters **n_adjacent** – Number of adjacent restriction fragments to remove

class ccanalyser.tools.filter.**TriCSliceFilter**(*slices*, *filter_stages=None*, ***sample_kwargs*)

Bases: *ccanalyser.tools.filter.CCSliceFilter*

Perform Tri-C slice filtering (inplace) and reporter identification.

SliceFilter tuned specifically for Tri-C data. Whilst the vast majority of filters are inherited from CCSliceFilter, this class has addtional methods for Tri-C analysis i.e. remove_slices_with_one_reporter. The default filtering order is:

- remove_unmapped_slices
- remove_slices_without_re_frag_assigned
- remove_orphan_slices
- remove_multi_capture_fragments
- remove_blacklisted_slices
- remove_non_reporter_fragments
- remove_multicapture_reporters
- remove_duplicate_re_frags
- remove_duplicate_slices
- remove_duplicate_slices_pe
- remove_non_reporter_fragments
- remove_slices_with_one_reporter

See the individual methods for further details.

slices
Annotated slices dataframe.

Type pd.DataFrame

fragments
Slices dataframe aggregated by parental read.

Type pd.DataFrame

reporters
Slices identified as reporters.

Type pd.DataFrame

filter_stages
Dictionary containg stages and a list of class methods (str) required to get to this stage.

Type dict

slice_stats
Provides slice level statistics.

Type pd.DataFrame

read_stats
Provides statistics of slice filtering at the parental read level.

Type pd.DataFrame

filter_stats
Provides statistics of read filtering.

Type pd.DataFrame

remove_slices_with_one_reporter()
Removes fragments if they do not contain at least two reporters.

class ccanalyser.tools.filter.**TiledCSliceFilter**(*slices*, *filter_stages=None*, ***sample_kwargs*)

Bases: *ccanalyser.tools.filter.SliceFilter*

Perform Tiled-C slice filtering (inplace) and reporter identification.

SliceFilter tuned specifically for Tiled-C data. This class has addtional methods to remove common artifacts in Tiled-C data i.e. non-capture fragments, multi-capture (with different tiled regions) fragments. A reporter is defined differently in a Tiled-C analysis as a reporter slice can also be a capture slice.

The default filter order is as follows:

- remove_unmapped_slices
- remove_orphan_slices
- remove_blacklisted_slices
- remove_non_capture_fragments
- remove_dual_capture_fragments
- remove_slices_without_re_frag_assigned
- remove_duplicate_re_frags
- remove_duplicate_slices
- remove_duplicate_slices_pe
- remove_orphan_slices

See the individual methods for further details.

slices
Annotated slices dataframe.

Type pd.DataFrame

fragments
Slices dataframe aggregated by parental read.

Type pd.DataFrame

reporters
Slices identified as reporters.

Type pd.DataFrame

filter_stages
Dictionary containg stages and a list of class methods (str) required to get to this stage.

Type dict

slice_stats
Provides slice level statistics.

Type pd.DataFrame

read_stats
Provides statistics of slice filtering at the parental read level.

Type pd.DataFrame

filter_stats
Provides statistics of read filtering.

Type pd.DataFrame

property fragments
Summarises slices at the fragment level.

Uses pandas groupby to aggregate slices by their parental read name (shared by all slices from the same fragment). Also determines the number of reporter slices for each fragment.

Returns Slices aggregated by parental read name.

Return type pd.DataFrame

property slice_stats
Statistics at the slice level.

Returns Statistics per slice.

Return type pd.DataFrame

property cis_or_trans_stats
Extracts reporter cis/trans statistics from slices.

Unlike Capture-C/Tri-C reporter slice can also be capture slices as all slices within the capture region are considered as reporters. To extract cis/trans statistics, one capture slice in each fragment is considered to be the "primary capture" this then enables merging of this "primary capture" with the other reporters both inside and outside of the tiled region.

Returns Reporter cis/trans statistics

Return type pd.DataFrame

remove_slices_outside_capture()
Removes slices outside of capture region(s)

remove_non_capture_fragments()
Removes fragments without a capture assigned

remove_dual_capture_fragments()
Removes a fragment with multiple different capture sites.

Modified for TiledC filtering as the fragment dataframe is generated slightly differently.

ccanalyser.tools.io module

class ccanalyser.tools.io.**FastqReaderProcess**(*input_files: Union[str, list], outq: multiprocessing.context.BaseContext.Queue, read_buffer: int = 100000, read_counter: Optional[multiprocessing.managers.BaseManager.register.<locals>.te = None, n_subprocesses: int = 1, statq: Optional[multiprocessing.context.BaseContext.Queue] = None*)

Bases: multiprocessing.context.Process

Reads fastq file(s) in chunks and places them on a queue.

input_file
Input fastq files.

outq
Output queue for chunked reads/read pairs.

statq
(Not currently used) Queue for read statistics if required.

read_buffer
Number of reads to process before placing them on outq

read_counter
(Not currently used) Can be used to sync between multiple readers.

n_subproceses
Number of processes running concurrently. Used to make sure enough termination signals are used.

run()
Performs reading and chunking of fastq file(s).

class ccanalyser.tools.io.**FastqReadFormatterProcess**(*inq: multiprocessing.context.BaseContext.SimpleQueue, outq: multiprocessing.context.BaseContext.SimpleQueue, formatting: Optional[list] = None*)

Bases: multiprocessing.context.Process

run()
Method to be run in sub-process; can be overridden in sub-class

class ccanalyser.tools.io.**FastqWriterSplitterProcess**(*inq: multiprocessing.context.BaseContext.Queue, output_prefix: Union[str, list], paired_output: bool = False, gzip=False, compression_level: int = 3, compression_threads: int = 8, n_subprocesses: int = 1, n_workers_terminated: int = 0, n_files_written: int = 0*)

Bases: multiprocessing.context.Process

run()
Method to be run in sub-process; can be overridden in sub-class

class ccanalyser.tools.io.**FastqWriterProcess**(*inq: multiprocessing.context.BaseContext.Queue*, *output: Union[str, list]*, *compression_level: int = 5*, *n_subprocesses: int = 1*)

Bases: multiprocessing.context.Process

run()
Method to be run in sub-process; can be overridden in sub-class

ccanalyser.tools.io.**parse_alignment**(*aln*)
Parses reads from a bam file into a list.

Extracts: -read name -parent reads -flashed status -slice number -mapped status -multimapping status -chromosome number (e.g. chr10) -start (e.g. 1000) -end (e.g. 2000) -coords e.g. (chr10:1000-2000)

Parameters **aln** – pysam.AlignmentFile.

Returns Containing the attributes extracted.

Return type list

ccanalyser.tools.io.**parse_bam**(*bam*)
Uses parse_alignment function convert bam file to a dataframe.

Extracts: -'slice_name' -'parent_read' -'pe' -'slice' -'mapped' -'multimapped' -'chrom' -'start' -'end' -'coordinates'

Parameters **bam** – File name of bam file to process.

Returns DataFrame with the columns listed above.

Return type pd.Dataframe

ccanalyser.tools.pileup module

class ccanalyser.tools.pileup.**CoolerBedGraph**(*cooler_fn: str*, *sparse: bool = True*, *only_cis: bool = False*)

Bases: object

Generates a bedgraph file from a cooler file created by interactions-store.

cooler
Cooler file to use for bedgraph production

Type cooler.Cooler

capture_name
Name of capture probe being processed.

Type str

sparse
Only output bins with interactions.

Type bool

only_cis
Only output cis interactions.

Type bool

property bedgraph
Returns: pd.DataFrame: DataFrame in bedgraph format.

property reporters
Interactions with capture fragments/bins.

Returns DataFrame containing just bins interacting with the capture probe.

Return type pd.DataFrame

normalise_bedgraph(*scale_factor=1000000.0*) → pandas.core.frame.DataFrame
Normalises the bedgraph.

Uses the number of cis interactions to normalise the bedgraph counts.

Parameters **scale_factor** (*int, optional*) – Scaling factor for normalisation. Defaults to 1e6.

Returns Normalised bedgraph formatted DataFrame

Return type pd.DataFrame

to_file(*fn: os.PathLike*, *normalise: bool = False*, ***normalise_kwargs*)
Outputs the bedgraph dataframe to a file.

If normalise is True, will also normalise the counts by the number of cis interactions.

Parameters

- **fn** (*os.PathLike*) – Output file name.
- **normalise** (*bool, optional*) – Normalise the bedgraph before writing to file. Defaults to False.

class ccanalyser.tools.pileup.**CoolerBedGraphWindowed**(*cooler_fn: str*, *binsize: int = 5000.0*, *binner: Optional[ccanalyser.tools.storage.CoolerBinner] = None*, *sparse=True*)
Bases: *ccanalyser.tools.pileup.CoolerBedGraph*

normalise_bedgraph(*scale_factor=1000000.0*)
Normalises the bedgraph.

Uses the number of cis interactions to normalise the bedgraph counts.

Parameters **scale_factor** (*int, optional*) – Scaling factor for normalisation. Defaults to 1e6.

Returns Normalised bedgraph formatted DataFrame

Return type pd.DataFrame

property reporters_binned

class ccanalyser.tools.pileup.**CCBedgraph**(*path=None*, *df=None*, *capture_name=''*, *capture_chrom=''*, *capture_start=''*, *capture_end=''*)
Bases: object

property score

property coordinates

to_bedtool()

to_file(*path*)

ccanalyser.tools.plotting module

class ccanalyser.tools.plotting.**CCMatrix**(*cooler_fn: os.PathLike*, *binsize: 5000*, *capture_name: str*, *remove_capture=False*)

Bases: object

get_matrix(*coordinates*, *field='count'*)

get_matrix_normalised(*coordinates*, *normalisation_method=None*, ***normalisation_kwargs*)

ccanalyser.tools.statistics module

class ccanalyser.tools.statistics.**DeduplicationStatistics**(*sample: str*, *read_type: str = 'pe'*, *reads_total: int = 0*, *reads_unique: int = 0*)

Bases: object

property df

class ccanalyser.tools.statistics.**DigestionStats**(*read_type*, *read_number*, *unfiltered*, *filtered*)

Bases: tuple

filtered
Alias for field number 3

read_number
Alias for field number 1

read_type
Alias for field number 0

unfiltered
Alias for field number 2

ccanalyser.tools.statistics.**collate_histogram_data**(*fnames*)

ccanalyser.tools.statistics.**collate_read_data**(*fnames*)

ccanalyser.tools.statistics.**collate_slice_data**(*fnames*)

ccanalyser.tools.statistics.**collate_cis_trans_data**(*fnames*)

ccanalyser.tools.statistics.**extract_trimming_stats**(*fn*)

ccanalyser.tools.storage module

ccanalyser.tools.storage.**get_capture_coords**(*viewpoint_file: str*, *viewpoint_name: str*)

ccanalyser.tools.storage.**get_capture_bins**(*bins*, *viewpoint_chrom*, *viewpoint_start*, *viewpoint_end*)

ccanalyser.tools.storage.**create_cooler_cc**(*output_prefix: str*, *bins: pandas.core.frame.DataFrame*, *pixels: pandas.core.frame.DataFrame*, *capture_name: str*, *capture_viewpoints: os.PathLike*, *capture_bins: Optional[Union[int, list]] = None*, *suffix=None*, ***cooler_kwargs*) → os.PathLike

Creates a cooler hdf5 file or cooler formatted group within a hdf5 file.

Parameters

- **output_prefix** (*str*) – Output path for hdf5 file. If this already exists, will append a new group to the file.
- **bins** (*pd.DataFrame*) – DataFrame containing the genomic coordinates of all bins in the pixels table.
- **pixels** (*pd.DataFrame*) – DataFrame with columns: bin1_id, bin2_id, count.
- **capture_name** (*str*) – Name of capture probe to store.
- **capture_viewpoints** (*os.PathLike*) – Path to capture viewpoints used for the analysis.
- **capture_bins** (*Union[int, list], optional*) – Bins containing capture viewpoints. Can be determined from viewpoints if not supplied. Defaults to None.
- **suffix** (*str, optional*) – Suffix to append before the .hdf5 file extension. Defaults to None.

Raises **ValueError** – Capture name must exactly match the name of a supplied capture viewpoint.

Returns Path of cooler hdf5 file.

Return type os.PathLike

class ccanalyser.tools.storage.**GenomicBinner**(*chromsizes: Union[os.PathLike, pandas.core.frame.DataFrame, pandas.core.series.Series]*, *fragments: pandas.core.frame.DataFrame*, *binsize: int = 5000*, *n_cores: int = 8*, *method: Literal[midpoint, overlap] = 'midpoint'*, *min_overlap: float = 0.2*)

Bases: object

Provides a conversion table for converting two sets of bins.

chromsizes
Series indexed by chromosome name containg chromosome sizes in bp

Type pd.Series

fragments
DataFrame containing bins to convert to equal genomic intervals

Type pd.DataFrame

binsize
Genomic bin size

Type int

min_overlap
Minimum degree of intersection to define an overlap.

Type float

n_cores
Number of cores to use for bin intersection.

Type int

property bins
Equal genomic bins.

Returns DataFrame in bed format.

Return type pd.DataFrame

property bin_conversion_table
Returns: pd.DataFrame: Conversion table containing coordinates and ids of intersecting bins.

class ccanalyser.tools.storage.**CoolerBinner**(*cooler_fn: os.PathLike*, *binsize: Optional[int] = None*, *n_cores: int = 8*, *binner: Optional[ccanalyser.tools.storage.GenomicBinner] = None*)

Bases: object

Bins a cooler file into equal genomic intervals.

cooler
(cooler.Cooler): Cooler instance to bin.

binner
Binner class to generate bin conversion tables

Type ccanalyser.storeage.GenomicBinner

binsize
Genomic bin size

Type int

scale_factor
Scaling factor for normalising interaction counts.

Type int

n_cis_interactions
Number of cis interactions with the capture bins.

Type int

n_cores
Number of cores to use for binning.

Type int

property bins
Returns: pd.DataFrame: Even genomic bins of a specified binsize.

property bin_conversion_table
Returns: pd.DataFrame: Conversion table containing coordinates and ids of intersecting bins.

property capture_bins
Returns: pd.DataFrame: Capture bins converted to the new even genomic bin format.

property pixel_conversion_table
Returns: pd.DataFrame: Conversion table to convert old binning scheme to the new.

property pixels
Returns: pd.DataFrame: Pixels (interaction counts) converted to the new binning scheme.

normalise_pixels(*n_fragment_correction: bool = True*, *n_interaction_correction: bool = True*, *scale_factor: int = 1000000.0*)
Normalises pixels (interactions).

Normalises pixels according to the number of restriction fragments per bin and the number of cis interactions. If both normalisation options are selected, will also provide a dual normalised column.

Parameters

- **n_fragment_correction** (*bool, optional*) – Updates the pixels DataFrame with counts corrected for the number of restriction fragments per bin. Defaults to True.
- **n_interaction_correction** (*bool, optional*) – Updates the pixels DataFrame with counts corrected for the number of cis interactions. Defaults to True.
- **scale_factor** (*int, optional*) – Scaling factor for n_interaction_correction. Defaults to 1e6.

to_cooler(*store*, *normalise=False*, ***normalise_options*)

ccanalyser.tools.storage.**link_bins**(*clr: os.PathLike*)
Reduces cooler storage space by linking "bins" table.

All of the cooler "bins" tables containing the genomic coordinates of each bin are identical for all cooler files of the same resoultion. As cooler.create_cooler generates a new bins table for each cooler, this leads to a high degree of duplication.

This function hard links the bins tables for a given resolution to reduce the degree of duplication.

Parameters **clr** (*os.PathLike*) – Path to cooler hdf5 produced by the merge command.

5.1.4 Utilities

ccanalyser.utils.**open_logfile**(*fn: str*) → IO
Handles instances where the log file is sys.stdout

ccanalyser.utils.**merge_dictionaries**(*dicts: list*) → dict
Merges multiple dictionary entries

ccanalyser.utils.**invert_dict**(*d: dict*) → dict
Inverts key: value pairs into value: key pairs

ccanalyser.utils.**is_on**(*param: str*) → bool
Returns True if parameter in "on" values

ccanalyser.utils.**is_off**(*param: str*)
Returns True if parameter in "off" values

ccanalyser.utils.**is_none**(*param: str*) → bool
Returns True if parameter is none

ccanalyser.utils.**get_human_readable_number_of_bp**(*bp: int*) → str
Converts integer into human readable basepair number

ccanalyser.utils.**is_valid_bed**(*bed: Union[str, pybedtools.bedtool.BedTool]*, *verbose=True*) → bool
Returns true if bed file can be opened and has at least 3 columns

ccanalyser.utils.**bed_has_name**(*bed: Union[str, pybedtools.bedtool.BedTool]*) → bool
Returns true if bed file has at least 4 columns

ccanalyser.utils.**bed_has_duplicate_names**(*bed*) → bool
Returns true if bed file has no duplicated names

ccanalyser.utils.**get_re_site**(*recognition_site: Optional[str] = None*) → str
Obtains the recogniton sequence for a supplied restriction enzyme or correctly formats a supplied recognition sequence.

Parameters

- **– DNA sequence to use for fasta digestion e.g. "GATC"** (*cut_sequence*) –
- **– Name of restriction enzyme e.g. DpnII** (*restriction_enzyme*) –

Returns recognition sequence e.g. "GATC"

Raises **ValueError** – Error if restriction_enzyme is not in known enzymes

ccanalyser.utils.**hash_column**(*col: Iterable*, *hash_type=64*) → list
Convinience function to perform hashing using xxhash on an iterable. Not vectorised.

ccanalyser.utils.**split_intervals_on_chrom**(*intervals: Union[str, pybedtools.bedtool.BedTool, pandas.core.frame.DataFrame]*) → dict
Creates dictionary from bed file with the chroms as keys

ccanalyser.utils.**intersect_bins**(*bins_1: pandas.core.frame.DataFrame*, *bins_2: pandas.core.frame.DataFrame*, ***bedtools_kwargs*) → pandas.core.frame.DataFrame
Intersects two sets of genomic intervals using bedtools intersect.

Formats the intersection in a clearer way than pybedtool auto names.

ccanalyser.utils.**load_json**(*fn*) → dict
Convinence function to load gziped json file using xopen.

ccanalyser.utils.**get_timing**(*task_name=None*) → Callable
Decorator: Gets the time taken by the wrapped function

class ccanalyser.utils.**NaturalOrderGroup**(*name=None*, *commands=None*, ***attrs*)
Bases: click.core.Group

Simple class to ensure subcommand order is maintained by click.

list_commands(*ctx*)
Returns a list of subcommand names in the order they should appear.

ccanalyser.utils.**zap_files**(*files*)
Runs cgatcore zap_files on all inputs

ccanalyser.utils.**get_ucsc_color**(*color*) → str
Converts rgb to UCSC compatable colours

ccanalyser.utils.**get_colors**(*items: Iterable*, *colors: Optional[Iterable] = None*)
Extracts the appropriate number of colours for the items and formats them for UCSC.

ccanalyser.utils.**make_group_track**(*bigwigs: list*, *key: Union[callable, str, int]*, *overlay=True*) → dict
Generates a UCSC super track by grouping inputs by the provided key.

class ccanalyser.utils.**PysamFakeEntry**(*name*, *sequence*, *quality*)
Bases: object

Testing class used to supply a pysam FastqProxy like object

ccanalyser.utils.**convert_to_bedtool**(*bed: Union[str, pybedtools.bedtool.BedTool, pandas.core.frame.DataFrame]*) → pybedtools.bedtool.BedTool
Converts a str or pd.DataFrame to a pybedtools.BedTool object

ccanalyser.utils.**convert_bed_to_dataframe**(*bed: Union[str, pybedtools.bedtool.BedTool, pandas.core.frame.DataFrame]*) → pandas.core.frame.DataFrame
Converts a bed like object (including paths to bed files) to a pd.DataFrame

ccanalyser.utils.**format_coordinates**(*coordinates: Union[str, os.PathLike]*) → pybedtools.bedtool.BedTool

Converts coordinates supplied in string format or a .bed file to a BedTool.

Parameters **coordinates** (*Union[str, os.PathLike]*) – Coordinates in the form chr:start-end or a path.

Raises **ValueError** – Inputs must be supplied in the correct format.

Returns BedTool object containing the required coordinates.

Return type BedTool

ccanalyser.utils.**convert_interval_to_coords**(*interval: Union[pybedtools.cbedtools.Interval, dict]*, *named=False*) → str

Converts interval object to standard genomic coordinates.

e.g. chr1:1000-2000

Parameters **interval** (*Union[pybedtools.Interval, dict]*) – Interval to convert.

Returns Genomic coordinates in the format chr:start-end

Return type str

CHAPTER

SIX

GLOSSARY

fragment The entire fragment of DNA that is sequenced. e.g.:

```
--R1-->
----------fragment-----
               <----R2--
```

slice The results of *in silico* restriction digest of a fragment. e.g.:

```
---------GATC--------GATC-------------
--slice1-    -slice2-    -slice3------
```

capture A slice that overlaps one of the supplied viewpoints. e.g.:

```
---capture-slice-----
-------viewpoint-----GATC--------------
```

reporter A mapped slice from the same fragment as a capture slice used to report an interaction between two the viewpoint and another genomic region. e.g.:

```
---fragment1-capture-slice-GATC--fragment1-slice2---
```

PYTHON MODULE INDEX

C

INDEX

Symbols

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

Z