Counter RNAseq Window Documentation

Release

Bertrand Néron

June 29, 2017

CONTENTS

CHAPTER

ONE

USER GUIDE

1.1 overview

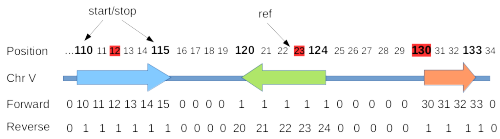
Counter **RNA** seq **W**indow is a package which aim to compute and visualize the coverage of RNA seq experiment.

The craw package contains two scripts *craw_coverage* and *craw_htmp*. *craw_coverage* compute the coverage, whereas *craw_htmp* allow to represent graphically the results of *craw_coverage* with a heat map.

1.1.1 craw_coverage:

craw_coverage take as input a bam file or wig file and an annotation file. **The annotation file** describe on which gene the craw_coverage must compute the coverage. The script compute a coverage for each position of this gene on a specified window around a position of reference on both sense and put the results on a matrix. The region of interest can be fixed for all genes (specified by the command line) or variable. In the this case the annotation file must contains two columns to specify beginning and the end of the region to take in account. The results in the matrix are centered on the position of reference of each gene. In the case of variable length of window the results are padded on left and right if necessary with *None* value. The results is saved in a file as a tabulated separated value by default with the same name as the bam file with the *.cov* extension (see *Outputs* for more details).

Below an example to illustrate how craw_coverage work. If we consider the following genome and we want to analyze 3 gene *foo*, *bar*, *buz*



On the figure above

- The first line represent the positions on the genome (1-based)
  - The bold position indicate the boundaries of region we want to analyse.
  - the red highlighted positions indicate, for each region, the position of reference.
- the second line represent the genes and their respective sense.
- the 2 last lines the coverage at each position of the genome for each strand

So to analyse these genes, we create an annotation file like following.

gene	Chr	strand	start	stop	ref
foo	V	+	110	115	112
bar	V	+	130	133	130
buz	V	-	120	124	123

the run a command line like:

```
craw_coverage --bam mygenome.bam --annot my_annotation --ref-col ref --start-col start --stop-col sto
```

will produce the following coverage matrix

Gene	sense	-2	-1	0	1	2	3
foo	S	10	11	12	13	14	15
foo	AS	1	1	1	1	1	1
bar	S	None	None	30	31	32	33
bar	AS	None	None	1	1	1	1
buz	AS	None	1	1	1	1	1
buz	S	None	24	23	22	21	20

1.1.2 craw_htmp:

craw_htmp read coverage file produced by *craw_coverage* and generate a graphical representation. It can produce either a file or an interactive graphic. The look and feel of the graphic and the format of supported outputs vary in function of the backend of matplotlib used (see *matplotlib configuration*). It can also produce raw images using pillow where 1 nucleotide is represent by 1 pixel.

1.1.3 Licensing

All files belonging to the **C**ounter **RNA**seq**W**indow (craw) package. are distributed under the GPLv3 licensing.

You should have received a copy of the GNU General Public License along with the package (see COPYING file). If not, see <http://www.gnu.org/licenses/>.

craw is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

craw is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

Authors: Bertrand Néron Copyright © 2017 Institut Pasteur (Paris). see COPYRIGHT file for details.

1.2 Installation

1.2.1 Requirements

For craw_coverage

- python > 3
- pysam >= 0.9.1.4

For craw_htmp

- python > 3
- pysam >= 0.9.1.4
- pandas >= 0.17.1
- numpy >= 1.11.2

- matplotlib >= 1.5.3
- pillow >= 3.4.2

1.2.2 Installation

Installation from package

Using pip

```
pip install craw
```

Do not forget to configure the *matplotlib* backend, specially if you use virtualenv. Otherwise on some platform there won't any output. See *matplotlib configuration* for more explanation.

Note: On MacOS install python > 3 from image on http://python.org . Then install craw using pip

```
pip3 install craw
```

craw will be installed in */Library/Framework/Python.Framework/Version/3.6/* So if you want to use directly craw_coverage and craw_htmp just create a symbolic linc like this:

```
ln -s /Library/Framework/Python.Framework/Version/3.6/bin/craw_coverage /usr/local/bin/craw_coverage
ln -s /Library/Framework/Python.Framework/Version/3.6/bin/craw_htmp /usr/local/bin/craw_htmp
```

The documentation (html and pdf) is located in /Library/Framework/Python.Framework/Version/3.6/share/craw/

Installation from repository

Clone the project and install with the setup.py

```
git clone https://gitlab.pasteur.fr/bneron/craw.git
cd craw
python3 setup.py install
```

Note: Instead of installing craw you can directly use the scripts from the repository. You can also use the package without installing it. To do this, you have to export the **CRAW_HOME** environment variable. *CRAW_HOME* must point to the *src* directory of the project. Then you can use *craw_coverage* and *craw_htmp* scripts located in *bin* directory.

This project is documented using sphinx. So if you use a clone, you have to generate the documentation from the source.

The project come from with some unit and functional tests. to test if everything work fine.

    cd $CRAW_HOME python3 tests/run_tests.py -vvv

matplotlib configuration

matplotlib is a python library to create graphics. *craw_htmp* use this library to generate heat map. The two parameters to configure for craw is:

- the backend
- figure.dpi

backend

matplolib lay on low level graphic library of youre computer. This library will determine the graphical formats manage by matplotlib and then by craw_htmp. Most of backend handle *png* but some library like *Qt* can handle 'jpeg', 'eps', *pdf* ...

In your *matplolibrc* file you must define the backend for instance to use Qt5

```
backend: qt5agg
```

An example of matplolibrc file and all supported backend is available here: http://matplotlib.org/users/customizing.html#a-sample-matplotlibrc-file

figure.dpi

It's not an essential option but *matplolib* and *craw_htmp* will produce better graphic (on screen) if you configure *matplotlib* to the native resolution of your screen. To know the resolution of your screen you can visit the following page https://www.infobyip.com/detectmonitordpi.php and report the resolution (for 1 inch) in *matplotlibrc* file like:

```
figure.dpi: 96
```

For full explanation on how to configure matplotlib read http://matplotlib.org/users/customizing.html#the-matplotlibrc-file.

1.3 Quick start

1.3.1 craw_coverage

craw_coverage need a file bam or wig to compute coverage and an annotation file to specify on which regions to compute these coverages.

- the -b or –bam allow to specify the path to the bam file.
- or alternatively the
  - -w, –wig option to specify the path to the wig file if the both strand aer encode in same file (negative value are on reverse strand)
  - –wig-for and –wig-rev to specify the paths to the wig files for the forward and reverse strand respectively
- the -a –annot allow to specify the path to the annotation file.

The –bam and –wig options are mutually exclusive but one of these option is required. –wig and –wig-for or –wig-rev are also mutually exclusive. the –annot option is required.

```
craw_coverage --bam ../WTE1.bam --annot ../annotations.txt --ref-col Position --before 100 --after 50
craw_coverage --wig ../WTE1.wig --annot ../annotations.txt --ref-col Position --before 100 --after 50
craw_coverage --wig-for ../WTE1_forward_strand.wig --wig-rev ../WTE1_reverse_strand.wig --annot ../an
--ref-col Position --before 100 --after 500
```

Warning: At the same place of *bam* file, there must be the corresponding index file (the *bam.bai* file). To generate the *.bai* file you have to use *samtools* program:

```
samtools index file.bam
```

see http://www.htslib.org/doc/ for more explanation.

with fix window

To compute the coverage on a fix window: we need to specify which column name in the annotation file define the reference position. The window will computed using this reference position.

- –ref-col

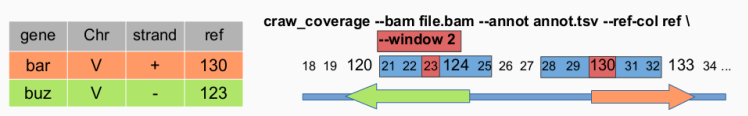
Note: if –ref-col is omitted craw_coverage will use the column position. If there not "position" column an error will occur.

---

two ways to determine the window:

with –window option for a window centered on the reference position.

- –window define the number of nucleotide to take in account before and after the reference position.

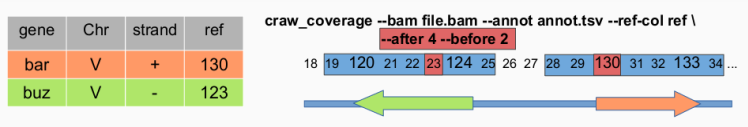



```
craw_coverage --bam ../WTE1.bam --annot ../annotations.txt --ref-col Position --window 100
```

This command will compute coverage using WTE1.bam and with annotations.txt file the column used to compute the window is 'Position' and the window length will be 100 nucleotide before the reference position and 100 nucleotides after (201 nucleotides length).

With an non centered window we have to specify two options *–before* and *–after*

- **–before BEFORE** define the number of nucleotide to take in account before the reference position.
- **–after AFTER** define the number of nucleotide to take in account after the reference position.




```
craw_coverage --bam ../WTE1.bam --annot ../annotations.txt --ref-col Position --before 100 --after 50
```

This command will compute coverage using WTE1.bam and with annotations.txt file the column used to compute the window is 'Position' and the window length will be 100 nucleotide before the reference position and 500 nucleotides after (201 nucleotides length).

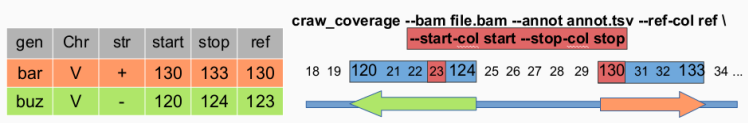
Note: –after and –before options must be set together and are incompatible with –window option.

---

with variable window

The regions must be specified in the annotation file.

- **–start-col COL** define the name of the column in annotation file which define the start position of the region to compute.
- **–stop-col COL** define the name of the column in annotation file which define the stop position of the region to compute.




```
craw_coverage --bam ../WTE1.bam --annot ../annotations.txt --ref-col annotation_start --start-col ann
```

This command will compute coverage using WTE1.bam and with annotations.txt file.

- The reference position will define by the *annotation_start* column
- The first nucleotide of the window will be define by *annotation_start* column.
- The last nucleotide of the window will be define by *annotation_end* column.

Other options

The following option are not mandatory:

- **-q QUAL_THR, –qual-thr QUAL_THR** The minimal quality of read mapping to take it in account. (default=15)
- **-s SUFFIX, –suffix SUFFIX** The name of the suffix to use for the output file. (default= *.cov*)
- **-o OUTPUT, –output OUTPUT** The path of the output (default= base name of annotation file with –suffix)
- **–version** display version information and quit.
- **–verbose, -v** increase the verbosity of the output (this option can be repeat several times as -vv).
- **–quiet** decrease verbosity of the output. By default craw_coverage is slightly verbose and display a progress bar. This option can be useful to disable any progression information on batch run.
- **-h –help** disply the inline help and exit.

Warning: by default craw_coverage use a quality threshold of 15 (like pysam)

Note: strand column mut named *strand* and can take *1/-1* or *+/- for/rev* as value for forward/reverse strands.

Warning: the coverage file can be huge depending on the number of gene to compute the coverage and the size of the window for instance for 6000 genes with a window of 15000 nt the cov file will weight almost 900Mb.

1.3.2 craw_htmp

Compute a figure from a file of coverage generated by *craw_coverage*. By default, display a figure with two heatmap one for the sense the other for the antisense. But it work also if the coverage file contains *sense* or *anti sense* data only.

Mandatory arguments

- **cov_file** The path to the coverage file (the output of).

Data options

- **–crop CROP CROP**: Crop the matrix. This option need two values the name of the first and last column to keep [start col, stop col] eg –crop -10 1000

```
craw_htmp --crop 0 2000 WTE1_var_window.cov
```

  This command will display only column '0' to '2000', included, of the matrix generated by craw_coverage.

- **–sort-using-col COL** sort the data using the column name 'COL' (descending).
- **–sort-using-file SORT_USING_FILE** sort the rows using a file. The file must have on the first line the name of the column to use for sorting and each line must match to a value contained in the matrix.
- **–sort-by-gene-size [start_col,stop_col [start_col,stop_col ...]]** The rows will be sorted by gene size using *start_col* and *stop_col* to compute length. *start_col* and *stop_col* must be a string separated by comma. If start_col and stop_col are not specify *annotation_start,annotation_end* will be used.
- **–sense-only** Display only sense matrix (default is display both).
- **–antisense-only** Display only anti sense matrix (default is display both).

Warning: Don't put the **–sort-by-gene-size** option without value as last option just before the coverage file. In this case the craw_htmp will don't work. If you want to use only this option, use the **-v** option after **–sort-by-gene-size**

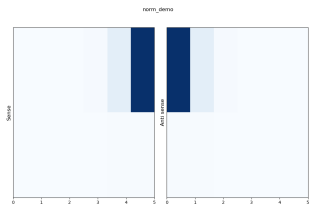
```
craw_htmp --sort-by-gene-size -v WTE1_0_2000.cov
```

Figure options

Normalization options

craw_htmp provide several methods to normalize data before to display them: below the different figures illustrate the result of each normalization methods on the following matrix x.

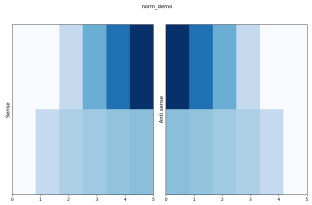
Pos	0	1	2	3	4	5
S	0	1	10	100	1000	10000
AS	10000	1000	100	10	1	0
S	1	10	20	30	40	50
AS	50	40	30	20	10	1



linear normalisation

–norm lin: a linear normalization is applied on the whole matrix.

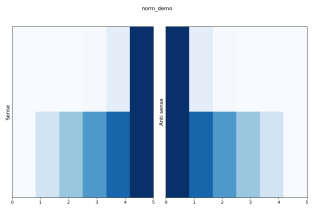
- x the original matrix
- x_i the value of a cell in the original matrix
- z_i the value of a cell in the normalized matrix
- $z_i = \frac{x_i - min(x)}{max(x) - min(x)}$



logaritmic normalisation

–norm log: a 10 base logarithm will be applied on the data before matrix normalization.

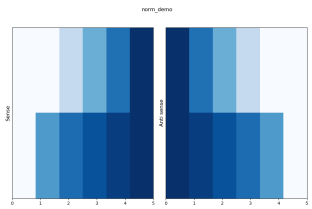
1. replace all 0 values by 1
2. $z_i = log_{10}(x_i)$
3. $w_i = \frac{z_i - min(z)}{max(z) - min(z)}$



linear normalisation row by row

–norm row: mean that a linear normalisation is compute row by row.

- x the original matrix
- x_{ij} the value of a cell in the original matrix with I rows and J columns
- x_i the values of the i row
- $z_{ij} = \frac{x_{ij} - min(x_i)}{max(x_i) - min(x_i)}$



logaritmic normalisation row by row

–norm log+row mean a 10 base logarithm will be applied before a normalisation row by row.

- x the original matrix
- x_{ij} the value of a cell in the original matrix with I rows and J columns
- x_i the values of the i row

1. replace all 0 values by 1
2. $z_{ij} = log_{10}(x_i j)$
3. $w_{ij} = \frac{z_{ij} - min(z_i)}{max(z_i) - min(z_i)}$

Note:

- **'row+log'** is an alias for **'log+row'**
- The default normalisation is **lin**

Other figure options

- **–cmap CMAP** The color map used to display data. The allowed values are defined in http:matplotlib.org/examples/color/colormaps_reference.html eg: Blues, BuGn, Greens, GnBu, ... (default: Blues).
- **–title TITLE** The figure title. It will display on the top of the figure. (default: the name of the coverage file without extension).
- **–dpi DPI** The resolution of the output (default=100).

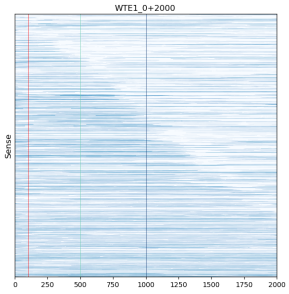
  This option work only if **–out** option is specified. To set the right dpi for screen displaying use the *matplotlib configuration* file.
- **–size SIZE** Specify the figure size

  The value must be widexheight[unit] or 'raw'. If value is 'raw' it will be produce two image files (for sense and antisense) with one pixel correspond to one coverage value. Otherwise, 'wide' and 'height' must be positive integers By default *unit* is in inches. eg:

  – 7x10 or 7x10in for 7 inches wide by 10 inches height.

  – 70x100mm for 70 mm by 100 mm.

  default=7x10 or 10x7 depending of the figure orientation (see layout).
- **–mark** POS <COLOR> will draw a vertical line at the position POS with the color <COLOR>



  COLOR can be the name of the most common html color red, yellow, ... or a value of a RGB in hexadecimal format like #rgb or #rrggbb for instance #ff0000 represent pure red. (don't forget to surround the hexadecimal color with quotes on commandline)

  If COLOR is omitted the color of the highest value of the color map used for the drawing will be used (The default color map is Blues).

```
craw_htmp --norm log \-\-mark 1000 \-\-mark 500 MediumAquamarine \-\-mark 100 "#ff0000" \-\-sens
```

  for list of HTML colors:

  – https://en.wikipedia.org/wiki/Web_colors

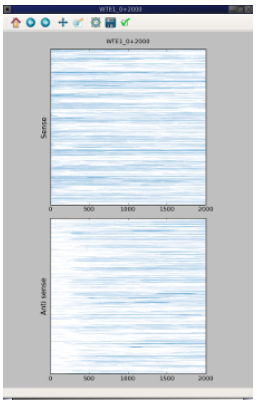
  – https://www.w3schools.com/colors/colors_names.asp

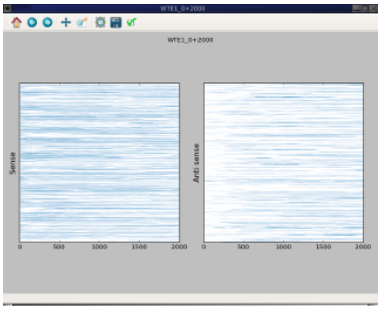
Warning: The –mark option must not be the last option on the command line (just before the coverage file), otherwise an error will occurred.:

```
craw_htmp --out my_fig.png --mark 10 red --mark 0  WTE1_0_2000.cov => raise an error
craw_htmp --mark 10 red --mark 0 --out my_fig.png WTE1_0_2000.cov => work
```

Layout options

- **–sense-on-left** Where to display the sense matrix relative to antisense matrix.
- **–sense-on-right** Where to display the sense matrix relative to antisense matrix.
- **–sense-on-top** Where to display the sense matrix relative to antisense matrix.
- **–sense-on-bottom** Where to display the sense matrix relative to antisense matrix.





The fisrt screen capture use *–sense-on-top* whereas the second capture used *–sense-on-left option*.

Note: default is top.

Other options

- **-h, –help** Display the help message and exit
- **–out OUT** The name of the file (the format will based on the extension) to save the figure. Instead of displaying the figure on the screen, save it directly in this file.

- **-v, –verbose** Increase output verbosity. By default craw_htmp is relatively quiet (display only warning and error), if you want to display also the processing step just add -v on the commandline (or -vv to display also the debugging message).
- **–version** Display version information and quit.

1.4 Inputs / Outputs

1.4.1 craw_coverage

Inputs

craw_coverage* need a file bam or wig to compute coverage and an annotation file to specify on which regions to compute these coverages.

bam file

craw_coverage can use a file of alignment reads called bam file. a bam file is a short DNA sequence read alignments in the Binary Alignment/Map format (.bam). *craw_coverage* needs also the corresponding index file (bai). The index file must be located beside the bam file with the same name instead to have the *.bam* extension it end by *.bai* extension. If you have not the index file you have to create it.

To index a bam file you need samtools. The command line is

```
samtools index file.bam
```

For more explanation see http://www.htslib.org/doc/ .

wig file

craw_coverage can compute coverage also from wig file see https://wiki.nci.nih.gov/display/tcga/wiggle+format+specification and http://genome.ucsc.edu/goldenPath/help/wiggle.html . for format specifications. Compare d to these specifications craw support coverages on both strands. the positive coverages scores are on the forward strand whereas the negative ones are on the reverse strand.

```
track type=wiggle_0 name="demo" color=96,144,246 altColor=96,144,246 autoScale=on  graphType=bar
variableStep chrom=chrI span=1
72      12.0000
73      35.0000
74      70.0000
75      127.0000
...
72      -88.0000
73      -42.0000
74      -12.0000
75      -1.0000
```

In the example above the coverage on the Chromosome I for the positions 72, 73, 74, 75 are 12, 35, 70, 127 on the forward strand and 88, 42, 12, 1 on the reverse strand.

annotation file

The annotation file is a *tsv* file by default. It's mean that it is a text file with value separated by tabulation (not spaces) or commas. But if a separator is specified (–sep) it can be a csv file or any columns file.

The first line of the file must be the name of the columns the other lines the values. Each line represent a row.

```
name    gene    chromosome      strand  Position
YEL072W RMD6    chrV    +       14415
YEL071W DLD3    chrV    +       17845
YEL070W DSF1    chrV    +       21097
YEL066W HPA3    chrV    +       27206
YEL065W SIT1    chrV    +       29543
YEL062W NPR2    chrV    +       36254
YEL058W PCM1    chrV    +       44925
YEL056W HAT2    chrV    +       48373
```

All lines starting with '#' character will be ignored.

```
# This is the annotation file for Wild type
# bla bla ...
name    gene    chromosome      strand  Position
YEL072W RMD6    chrV    +       14415
YEL071W DLD3    chrV    +       17845
YEL070W DSF1    chrV    +       21097
YEL066W HPA3    chrV    +       27206
YEL065W SIT1    chrV    +       29543
YEL062W NPR2    chrV    +       36254
YEL058W PCM1    chrV    +       44925
YEL056W HAT2    chrV    +       48373
```

mandatory columns There is 3 mandatory columns in the annotation file.

columns with fixed name two with a fixed name:

- **strand** indicate on which strand is located the region of interest. The authorized values for this columns are +/- , 1/-1 or for/rev.
- **chromosome** the chromosome name where is located the region of interest.

columns with variable name In addition of these two columns the column to define the position of reference is mandatory too, but the name of this column can be specified by the user. If it's not craw_coverage will use a column name 'position'.

If we want to compute coverage on variable window size, 2 extra columns whose name must be specified by the user by the following option:

- –start-col to define the beginning of the window (this position is included in the window)
- –stop-col to define the end of the window (this position is included in the window)

```
name    gene    type    chromosome      strand  annotation_start        annotation_end  has_transcrip
YEL072W RMD6    gene    chrV    1       13720   14415   1       14745   13569
YEL071W DLD3    gene    chrV    1       16355   17845   1       17881   16177
YEL070W DSF1    gene    chrV    1       19589   21097   1       21197   19539
YEL066W HPA3    gene    chrV    1       26721   27206   1       27625   26137
YEL065W SIT1    gene    chrV    1       27657   29543   1       29601   27625
YEL062W NPR2    gene    chrV    1       34407   36254   1       36401   34321
```

```
YEL058W PCM1    gene    chrV    1       43252   44925   1       44993   43217
YEL056W HAT2    gene    chrV    1       47168   48373   1       48457   47105
YEL052W AFG1    gene    chrV    1       56571   58100   1       58105   56537
```

```
craw_coverage --wig file.wig --annot annot.txt --ref-col annotation_start --start-col annotation_sta
```

The position of reference must be between start and end. The authorized values are positive integers.

Note: the position of reference can be used to define the reference and the start ot the end of the window.

```
craw_coverage --bam file.bam --annot annot.txt --ref-col annotation_start --start-col annotation_sta
```

All other columns are not necessary but will be reported as is in the coverage file.

Outputs

coverage_file

It's a *tsv* file with all columns found in annotation file plus the result of coverage position by position centered on the reference position define for each line. for instance

```
craw_coverage --wig=../data/small.wig --annot=../data/annotations.txt
--ref-col=annotation_start --before=0  --after=2000
```

In the command line above, the column '0' correspond to the annotation_start position the column '1' to annotation_start + 1 on so on until '2000' (here we display only the first 3 columns of the coverage).

```
# Running Counter RnAseq Window craw_coverage
# Version: craw NOT packaged, it should be a development version | Python 3.4
# Using: pysam 0.9.1.4 (samtools 1.3.1)
#
# craw_coverage run with the following arguments:
# --after=3
# --annot=../data/annotation_wo_start.txt
# --before=5
# --chr-col=chromosome
# --output=small_wig.cov
# --qual-thr=0
# --quiet=1
# --ref-col=Position
# --sense=mixed
# --sep=
# --strand-col=strand
# --suffix=cov
# --verbose=0
# --wig=../data/small.wig
sense   name    gene    type    chromosome      strand  annotation_start        annotation_end  has_t
S       YEL072W RMD6    gene    chrV    +       13720   14415   1       14745   13569   7       7
AS      YEL072W RMD6    gene    chrV    +       13720   14415   1       14745   13569   0       0
S       YEL071W DLD3    gene    chrV    +       16355   17845   1       17881   16177   31      33
```

The line starting with '#' are comments and will be ignored for further processing. But in traceability/reproducibility concern, in the comments *craw_coverage* indicate the version of the program and the arguments used for this experiment.

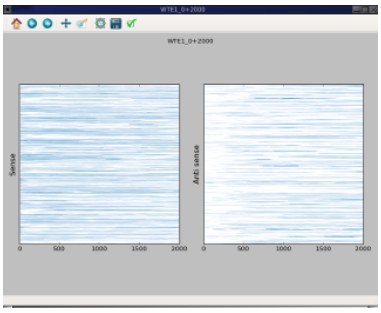
1.4.2 craw_htmp

Inputs

see *cov_out*

Outputs

The default output of *craw_htmp* (if –out is omitted) is grapical window on the screen. The figure display on the screen can be saved using the window menu.



It is also possible to generate directly a image file in various format by specifying the –out option. The output format will be deduced form the filename extension provide to –out option.

```
--out foo.jpeg  for jpeg image or --out foo.png  for png image
```

The supported format vary in function of the matloblib backend used (see *matplotlib configuration*).

If –size raw is used 2 files will be generated one for the sense and the other for the antisense. If –out is not specified it will be the name of the coverage file without extension and the format will be png.

```
craw_htmp foo_bar.cov --size raw
```

will produce *foo_bar.sense.png* and *foo_bar.antisense.png*

```
craw_htmp foo_bar.cov --size raw --out Xyzzy.jpeg
```

will produce *Xyzzy.sense.jpeg* and *Xyzzy.antisense.jpeg*

CHAPTER

TWO

DEVELOPER GUIDE

2.1 Overview

Scripts are located in *bin* directory, and use some modules located in *craw* directory.

- *craw_coverage* use module `craw.annotation` to handle annotation file and module `craw.coverage` to compute coverage this module rely on pysam.
- *craw_htmp* read coverage file generate by *craw_coverage* and produce graphical representation of data. This script use functions in module `craw.heatmap` in the form of heatmap. The module `craw.heatmap` have some capabilities to sort, crop, normalize data before represent them. this module rely on numpy, pandas to manipulate data (`craw.heatmap.sort`, `craw.heatmap.crop_matrix`, `craw.heatmap.lin_norm`, ...) and matplotlib and/or pillow to generate images (`craw.heatmap.draw_heatmap` , `craw.heatmap.draw_raw_image`)

2.2 reference API

2.2.1 craw

The *_get_version_message* is a private function that provide a human readable version of *craw* package and *python* which is common to all scripts. Each script have a public function *get_version_message* that call this function for the common part and add the version of all dependencies need for the script.

craw.**get_version_message**()

Returns A human readable version of the craw package version

Return type string

craw.**init_logger**(*log_level*)

Initiate the "root" logger for craw library all logger create in craw package inherits from this root logger This logger write logs on sys.stderr

Parameters **log_level** (*int*) – the level of the logger

2.2.2 annotation

The annotation module contains everything that is needed to parse annotation file and handle it.

AnnotationParser

The entry point to parse an annotation file is the *craw.annotation.AnnotationParser*. An annotation parser have two methods:

- *craw.annotation.AnnotationParser.get_annotations()* create a new type of Entry and iterate over the annotation file and for each line return a new instance of the newly *craw.annotation.Entry* class it just create on the fly.
- the other more technique give the maximum of nucleotides before and after the reference. It is needed to compute the size of the resulting matrix.

The force of this approach is to generate a new type of entry for each parsing. So it's very flexible and allow to fit with most of annotation file. But for one file, all the parsing use the same *Entry* class so it ensure the coherence in data.

new_entry_type

Is a factory which generate a new subclass of *craw.annotation.Entry* given the fields gather form the annotation file header (first line non starting with #) and the columns semantic given by the user. The first role of this factory is to check if all parameter given by user correspond ot header and do some coherence checking. If everything seems Ok it generate on the fly a new subclass of *craw.annotation.Entry*.

Entry Class

An Entry correspond to one line of the annotation file.

The Entry convert values if necessary (*strand* in a internal representation +/-, *position* in integer ...). It also expose a generic api to access some fields whatever the named of the columns.

annotation API reference

class craw.annotation.**AnnotationParser**(*path*, *ref_col*, *strand_col='strand'*, *chr_col='chromosome'*, *start_col=None*, *stop_col=None*, *sep='t'*)

Parse the annotation file

- create new type of Entry according to the header
- create one Entry object for each line of the file

__init__(*path*, *ref_col*, *strand_col='strand'*, *chr_col='chromosome'*, *start_col=None*, *stop_col=None*, *sep='\t'*)

Parameters

- **path** (*string*) – the path to the annotation file to parse.
- **ref_col** (*string*) – the name of the column for the reference position
- **chr_col** (*string*) – the name of the column for the chromosome
- **strand_col** (*string*) – the name of the column for the strand
- **start_col** (*string*) – the name of the column for start position
- **stop_col** (*string*) – the name of the column for the stop position
- **sep** (*string*) – The separator tu use to split fields

__weakref__
list of weak references to the object (if defined)

get_annotations()
Parse an annotation file and yield a *Entry* for each line of the file.

Returns a generator on a annotation file.

max()

Returns the maximum of bases to take in count before and after the reference position.

Return type tuple of 2 int

class craw.annotation.**Entry**(*values*)
Handle one entry (One line) of annotation file

__init__(*values*)

Parameters **values** (*list of string*) – the values parsed from one line of the annotation file

__weakref__
list of weak references to the object (if defined)

_convert(*field*, *value*)
Convert field parsed from annotation file in Entry internal value

Parameters

- **field** (*string*) – the field name associated to the value.
- **value** (*string*) – the value to convert

Returns the converted value

Return type any

Raise RuntimeError or value Error if a value cannot be converted

_switch_start_stop()
Switch start and stop value if self.start > self.stop This situation can occur if annotation regards the reverse strand

chromosome
The name of the Chromosome

header
The header of the annotation file

ref
The position of reference

start
The Position to start the coverage computation

stop
The position to end the coverage computation (included)

strand
the strand +/-

class craw.annotation.**Idx**(*col_name*, *idx*)

__getnewargs__()
Return self as a plain tuple. Used by copy and pickle.

static __new__ (*_cls*, *col_name*, *idx*)
Create new instance of Idx(col_name, idx)

__repr__ ()
Return a nicely formatted representation string

_asdict ()
Return a new OrderedDict which maps field names to their values.

classmethod _make (*iterable*, *new=<built-in method __new__ of type object>*, *len=<built-in function len>*)
Make a new Idx object from a sequence or iterable

_replace (*_self*, ***kwds*)
Return a new Idx object replacing specified fields with new values

col_name
Alias for field number 0

idx
Alias for field number 1

craw.annotation.**new_entry_type** (*name*, *fields*, *ref_col*, *strand_col='strand'*, *chr_col='chromosome'*, *start_col=None*, *stop_col=None*)
From the header of the annotation line create a new Entry Class inherited from Entry Class

Parameters

- **name** (*str*) – The name of the new class of entry.
- **fields** (*list of string*) – The fields constituting the new type of entry.
- **ref_col** (*string*) – The name of the column representing the position of reference (default is 'position').
- **strand_col** (*string*) – The name of the column representing the strand (default is 'strand').
- **chr_col** (*string*) – The name of the column representing the name of chromosome (default is 'chromosome').
- **start_col** (*string*) – The name of the column representing the position of the first base to compute the coverage (inclusive).
- **stop_col** (*string*) – The name of the column representing the position of the last base to compute the coverage (inclusive).

Returns a new class child of *Entry* which is able to store information corresponding to the header.

2.2.3 wig

This module allow to parse wig files (wig file specifications are available here: https://wiki.nci.nih.gov/display/tcga/wiggle+format+specification, http://genome.ucsc.edu/goldenPath/help/wiggle.html). The wig file handle by this modules slightly differ fom de canonic specifications as it allow to specify coverage on forward and reverse strand. If the coverage score is positive that mean that it's on the forward strand if it's negative, it's on the reverse strand.

The WigParser and helpers

The *craw.wig.WigParser* allow to parse the wig file. It read the file line by line, test the category of the line trackLine, declarationLine or dataLine and call the right method to parse the line and build the genome object.

The classes *craw.wig.VariableChunk* and *craw.wig.FixedChunk* are not keep in the final data model, they are just used to parsed the data lines and convert the wig file information (step, span) in coverages for each positions.

The data model to handle the wig information

The *craw.wig.Genome* objects contains *craw.wig.Chromosome* (each chromosomes ar unique and the names of chromosomes are unique). Each chromosomes contains the coverage for the both strands. To get the coverage for region or a position just access it with indices or slices as traditional python list, tuple, on so on. The slicing return two lists. The first list correspond to the coverage on this particular region for the forward strand, the second element for the reverse strand. By default the chromosomes are initialized with 0.0 as coverage for all positions.

All information specified in the track line are stored in the `infos` attribute of *craw.wig.Genome* as a dict.

wig API reference

class craw.wig.**Chromosome**(*name*, *size=1000000*)

Handle chromosomes. A chromosome as a name and contains *Chunk* objects (forward and reverse)

__getitem__(*pos*)

Parameters **pos** – a position or a slice (0 based) if pos is a slice the left indice is excluded

Returns the coverage at this position or corresponding to this slice.

Return type a list of 2 list of float [[float,...],[float, ...]]

Raises **IndexError** if pos is not in coverage or one bound of slice is out the coverage

__init__(*name*, *size=1000000*)

Parameters

- **name** (*str*) –
- **size** (*the default size of the chromosome. Each time we try to set a value greater than the chromosome the chromosome size is doubled. This is to protect the machine against memory swapping if the user provide a wig file with very big chromosomes.*) –

__len__()

Returns the actual length of the chromosome

Return type int

__setitem__(*pos*, *value*)

Parameters

- **pos** (int or `slice` object) – the postion (0-based) to set value
- **value** (*float or iterable of float*) – value to assign

Raises

- **ValueError** – when pos is a slice and value have not the same length of the slice
- **TypeError** – when pos is a slice and value is not iterable
- **IndexError** – if pos is not in coverage or one bound of slice is out the coverage

__weakref__

list of weak references to the object (if defined)

_estimate_memory(*col_nb*, *mem_per_col*)

Parameters

- **col_nb** (*int*) – the number of column of the new array or the extension

- **mem_per_col** (*int*) – the memory needed to create or extend an array with one col and 2 rows fill with 0.0

Returns the estimation of free memory available after creating or extending chromosome

Return type int

_extend(*size=1000000*, *fill=0.0*)

Extend this chromosome of the size size and fill with fill. :param size: the size (in bp) we want to increase the chromosome. :type size: int :param fill: the default value to fill the chromosome. :type fill: float or nan :raise MemoryError: if the chromosome extension could overcome the free memory.

class craw.wig.**Chunk**(***kwargs*)

Represent the data following a declaration line. The a Chunk contains sparse data on coverage on a region of one chromosomes on both strand plus data contains on the declaration line.

__init__(***kwargs*)

Parameters **kwargs** (*dictionary*) – the key,values pairs found on a Declaration line

__weakref__

list of weak references to the object (if defined)

is_fixed_step()

This is an abstract methods, must be implemented in inherited class :return: True if i's a fixed chunk of data, False otheweise :rtype: boolean

parse_data_line(*line*, *chrom*, *strand_type*)

parse a line of data and append the results in the corresponding strand This is an abstract methods, must be implemented in inherited class.

Parameters

- **line** (*string*) – line of data to parse (the white spaces at the end must be strip)
- **chrom** (*Chromosome* object.) – the chromosome to add coverage data
- **strand_type** (*string '+', '-', 'mixed'*) – which kind of wig is parsing: forward, reverse, or mixed strand

class craw.wig.**FixedChunk**(***kwargs*)

The FixedChunk objects handle data of 'fixedStep' declaration line and it's coverage data

is_fixed_step()

Returns True

Return type boolean

parse_data_line(*line*, *chrom*, *strand_type*)

parse line of data following a fixedStep Declaration. add the result on the corresponding strand (forward if coverage value is positive, reverse otherwise) :param line: line of data to parse (the white spaces at the end must be strip) :type line: string :param chrom: the chromosome to add coverage data :type chrom: *Chromosome* object. :param strand_type: which kind of wig is parsing: forward, reverse, or mixed strand :type strand_type: string '+' , '-', 'mixed'

class craw.wig.**Genome**

A genome is made of chromosomes and some metadata, called infos

__delitem__(*name*)

remove a chromosome from this genome

Parameters **name** (*string*) – the name of the chromosome to remove

Returns None

__getitem__(*name*)

Parameters **name** (*string*) – the name of the chromosome to retrieve

Returns the chromosome corresponding to the name.