HiCat: User Guide

Marc W. Schmid, marcschmid@gmx.ch

November 25, 2014

Contents

1 Installation

This section describes how to obtain and install HiCat. Source code, 64-bit binaries for Linux, Windows, and Mac, and R-Scripts can be downloaded on github.com/MWSchmid/HiCat. Even though the program can in principle run using little memory, the R-code can easily use several Gb of RAM. It is therefore strongly recommended to use a 64-bit system with at least 6 Gb of RAM.

1.1 Using pre-compiled binaries

If you have a 64 bit (Ubuntu-like) Linux, Windows (7) or MacOSX, use the pre-compiled binary. The binaries were built on Kubuntu 12.04, Windows 7, and MacOS 10.9 (versions below 10.9 were not tested). If you encounter problems with the binaries, try building the program from source (see section 4) and send a report to marcschmid@gmx.ch.

1.1.1 Linux

Download and unpack the archive `linux_64bit.zip`. Start HiCat directly either by double-clicking on it or from the terminal (you may need to make it executable first, right-click on the binary, open the "properties" dialog and check the box for "is executable" - or in a terminal type `chmod 755 filename`).

1.1.2 Windows

Download and unpack the archive `windows_64bit.zip`. Start the application directly by double-clicking on it.

1.1.3 Mac

Download and unpack the archive `mac_64bit.zip`. Mount the `*.dmg` file (double-click) and start the application by double-clicking on it. We do not recommend to use MacOSX 10.10, as it seems to have some problems with the memory allocation.

1.1.4 R-dependencies

HiCat requires the R libraries "gplots", "randomizeBE", and "MASS". You can install them with `install.packages(c("gplots", "randomizeBE", "MASS"))`

2 Step-by-step example for pre-processing Hi-C data

This section provides a step-by-step tutorial on how to get the tables for Hi-C data analysis in R starting from initial read files. The example data are from *A. thaliana* and comprise five seedling samples (two wild-types and three mutant samples) [1, 2]. Download and unpack the archive `At_pre-process_tutorial.zip` from github.com/MWSchmid/HiCat. It contains a folder with the *A. thaliana* reference genome, its annotation in gff format (TAIR10 from www.arabidopsis.org), and a few additional tracks (genomic sequencing, RNA-Seq and DNA methylation data). It additionally contains pre-processed `.bam` files for two Hi-C samples in case you would like to skip the download and alignment part of the tutorial (in this case go to section 2.4). The short reads download and alignment part is written for an Ubuntu-like Linux.

2.1 OPTIONAL: Installation of additional programs

Additional programs are required to download and align the short reads. It is later assumed that these programs reside in a folder that is included in your PATH environment variable. This can be done by either moving the programs into one of the by-default included folders (e.g. /usr/local/bin), or by adding the folder containing the programs to the PATH environment variable. Note that the latter is a temporary solution (the commands have to be entered each time you start a new terminal). Code for both options is given for each of the programs (note that the hash-tag `#` stands for comments, which do not have to be typed into the terminal).

- SRA toolkit
  Visit `www.ncbi.nlm.nih.gov/Traces/sra/sra.cgi?view=software`, download the archive for "Ubuntu Linux 64 bit architecture", unpack it, open a terminal, and type (adjust the path and version number):

  ```
  # SOLUTION 1
  cd /path/to/sratoolkit.x.x.x-x-ubuntu64/bin
  sudo cp -r * /usr/local/bin
  # SOLUTION 2 (temporary!)
  export PATH="$PATH:/path/to/sratoolkit.x.x.x-x-ubuntu64/bin"
  ```

- Subread [3]
  Visit `subread.sourceforge.net` and obtain the latest version. Follow the link in the box on the right side of the page, download the archive for linux, unpack it, open a terminal, and type (adjust the path and version number):

  ```
  # SOLUTION 1
  cd /path/to/subread-x.x.x-p1-Linux-x86_64/bin
  sudo cp -r * /usr/local/bin
  # SOLUTION 2 (temporary!)
  export PATH="$PATH:/path/to/subread-x.x.x-p1-Linux-x86_64/bin"
  ```

- SAMtools [4]
  Visit `sourceforge.net/projects/samtools/files/samtools/` and obtain the latest version. Download the archive and unpack it. SAMtools needs to be built from source. For this, install zlib (`zlib1g`, `zlib1g-dev`, and `zlib1g-dev` from the package manager), open a terminal, and type (adjust the path and version number):

  ```
  # COMPULSORY - BUILD INSTRUCTIONS
  cd /path/to/samtools-x.x.x
  make
  # SOLUTION 1
  cd /path/to/samtools-x.x.x
  sudo cp samtools /usr/local/bin
  # SOLUTION 2 (temporary!)
  export PATH="$PATH:/path/to/samtools-x.x.x"
  ```

2.2 OPTIONAL: Obtaining the short read data

The data used in this tutorial can be conveniently retrieved from NCBI using the SRA toolkit. Open a terminal to download the example data (takes several hours) in the working directory (e.g. At_pre-process_tutorial, which has been automatically created by unpacking the archive `At_pre-process_tutorial.zip`):

```
cd /path/to/At_pre-process_tutorial
fastq-dump --split-3 SRR1197490
fastq-dump --split-3 SRR1197491
fastq-dump --split-3 SRR1197492
fastq-dump --split-3 SRR681003
fastq-dump --split-3 SRR681004
```

The option `--split-3` ensures that the forward and reverse reads are written in separate `.fastq` files. Downloading one sample (e.g. SRR1197490) will therefore result in two `.fastq` files (SRR1197490_1.fastq and SRR1197490_2.fastq)

2.3 OPTIONAL: Aligning the short reads to the reference genome

In this tutorial, we use Subread [3] to align the reads to the reference genome. This requires a special index of the reference genome. Build this index with:

```
cd /path/to/At_pre-process_tutorial
subread-buildindex -o At_GI.nix TAIR10.fasta
```

You can now align the reads with Subread. Note that the option `-T 4` tells the computer to use four cores. You may need to change this according to your system. The option `--trim3` trims the reads to 50 bp, `-I 0` disables InDel detection, and -u allows only for unique alignments. The backslash in the code below indicates that all should be written on one single line. For each sample, the forward and reverse (later on, we use the term read-end as synonym for one read of a pair) need to be aligned separately to the reference genome (as the aligners normally expect both reads of a pair to align close to each other - which is not the case for Hi-C data).

```
cd /path/to/At_pre-process_tutorial

# the following three samples have 100 bp reads, so we can trim 50 bp
for SAMPLE in SRR1197490 SRR1197491 SRR1197492
do echo "$SAMPLE"
subread-align -u --trim3 50 -I 0 -T 4 -i At_GI.nix -r "${SAMPLE}_1.fastq" \
--BAMoutput -o "${SAMPLE}_1.bam"
subread-align -u --trim3 50 -I 0 -T 4 -i At_GI.nix -r "${SAMPLE}_2.fastq" \
--BAMoutput -o "${SAMPLE}_2.bam"
done

# the following two samples have only 50 bp reads, so we don't trim
for SAMPLE in SRR681003 SRR681004
do echo "$SAMPLE"
subread-align -P 6 -u -I 0 -T 4 -i At_GI.nix -r "${SAMPLE}_1.fastq" \
--BAMoutput -o "${SAMPLE}_1.bam"
subread-align -P 6 -u -I 0 -T 4 -i At_GI.nix -r "${SAMPLE}_2.fastq" \
--BAMoutput -o "${SAMPLE}_2.bam"
done
```

Some general notes on the alignment part. Any aligner should work with HiCat. However, the output may have to be reformatted into BAM (`.bam`), which is required by HiCat (if sorted or not does not matter). In cases where one sample was sequenced on multiple lanes (i.e. has multiple sequencing runs), it is recommended to process the runs individually and only combine them later (i.e. at the stage where they are loaded into R). An important point to consider is the length of the individual reads: The longer a read, the more likely it is that the read spans the restriction site where the two restriction fragments were ligated. A read spanning this site consists of sequences from both fragments and can thus not be aligned. Long reads should therefore be trimmed (especially at the 3' end) to ensure a high data recovery. It is also possible to align the reads iteratively (e.g. 100 bp, 75 bp, 50 bp, 25 bp). However, we tried the iterative mapping procedure proposed in HiClib [5] (reads trimmed to 100, 82, 64, 46, and 28 bp

and aligned with bowtie2 in --sensitive instead of --very-sensitive mode), and did not observe a substantial increase in successfully aligned read-pairs compared to the single-step approach with Subread. At least for the small, non-repetitive genome of *Arabidopsis*, alignment with Subreads thus seems more convenient, considering that it is 10 to 20 times faster than the iterative procedure with bowtie2.

2.4 *HiCat*: Hi-C data pre-processing

2.4.1 Merge reads

Up to now, the individual reads of the read-pairs have been processed separately. To see which regions in the genome interact with each other, the read-ends need to be merged into read-pairs again. To merge the two alignment (`.bam`) files, start HiCat and go to the "merge reads" tab. If you followed the alignment part above, specify for each sample (e.g. SRR1197490) the two `.bam` files (e.g. SRR1197490_1.bam and SRR1197490_2.bam) and a plain `.txt` file (e.g. SRR1197490_read_pairs.txt). Otherwise take the two samples located in the archive `At_pre-process_tutorial.zip` (WT_1/2.bam and morc_1/2.bam) and merge the reads for each sample (e.g. WT_read_pairs.txt and morc_read_pairs.txt). To start merging the read-ends, press `merge files`. Note, however, that the maximal number of reads stored in memory at once is per default set to a high value (1 billion). If RAM is limiting, this value may be adjusted (10 million reads require around 1.3 Gb of RAM). Start the procedure for all samples (the jobs will be added to a queue and processed after each other - all current jobs in the queue are listed in the "overview" tab). Merging the read-ends should not take too long. The program can process around 12.6 million read-end alignments per minute (corresponds to around 5.4 million successfully merged pairs per minute) on our test-system [6].

The "read-pair table" with the merged read pairs has six tab-separated columns: `chromA, strandA, posA, chromB, strandB, posB` and refer to the chromosome, strand, and position of an aligned read-end (A for the forward read, B for the reverse read). Note that the positions are zero-based (means that the first base on a chromosome is number 0).

2.4.2 Create fragments and bins

While the read-ends are being merged, we can create the table with the genomic fragments. Go to the "create fragments" tab, specify the reference genome (`TAIR10.fasta`), and a plain `.txt` file which will store the genomic fragments. Fragments are created based on either the restriction pattern or the fixed bin-sizes. All samples have been generated using the *Hind*III restriction enzyme. To create a table with the restriction fragments (e.g. fragments_HindIII.txt), specify the restriction site (AAGCTT) and press `create fragments`. In addition to the restriction fragments, create a table with 100 kb bins (e.g. fragments_100kb.txt, specify as 100'000 bp!). The bin-size corresponds to the resolution later on, so you may try different sizes as well. Note that by pressing `create fragments` the job is added to the queue as well. As a consequence, the table may not be produced instantly.

The "fragment tables" have four tab-separated columns: `fragmentNumber, chrom, start, end`. The tables are required for the mapping procedure and for the addition of additional experiments. Note that multiple restriction enzymes can be supplied as `patternA,patternB` (e.g. `AAGCTT,CCATGG` for *Hind*III and *Nco*I).

2.4.3 Create organism-specific R-code

In the meantime, we can as well generate the organism-specific R-code. For the data analysis in R, we require some basic information on the genome (i.e. the chromosome names, sizes, and number of restriction fragments). A file holding this information can be automatically created under the tab "create organism-specific R-code". Simply specify the reference genome (`TAIR10.fasta`), an R-source file (with the ending `.R`, e.g. HiCat-A-thaliana-TAIR10.R), the restriction site (AAGCTT), and press `create R-code`. Note that this organism-specific R-script also contains a function which defines the chromosomes you would like to consider during the analysis. The following chromosome identifiers (i.e. the header in the fasta file) are per default considered to be irrelevant by HiCat: `Mt, Pt, MtDNA, PtDNA, ChrM, ChrC, M, C, MT, CP, PT, Un`. Before you start the analysis in R, make sure that the function contains all chromosomes which you would like to analyze (the function is called `f.get.relevant.chromosomes()`).

2.4.4 Map read-pairs to fragments

Go to the "overview" tab and wait until all jobs have been processed (it would be possible though to continue directly and specify the files manually using their future path). We now assign/map the read-ends to the genomic fragments. Go to the "map read-pairs to fragments" tab, specify the fragment table (e.g. fragments_HindIII.txt or fragments_100kb.txt, see 2.4.2), the read-pair table (e.g. SRR1197490_read_pairs.txt or WT_read_pairs.txt, see 2.4.1), and a plain `.txt` file which will store the mapped read-pairs (e.g. SRR1197490_read_pairs_mapped.txt or WT_read_pairs_mapped.txt). For data analysis in R, this table can further be simplified: enable the "reduced matrix for R" and specify a corresponding plain `.txt` file (e.g. SRR1197490_100kb.txt or WT_100kb.txt).

The read-pairs can optionally be filtered using the approach proposed by Jin *et al.* [7]. Read-pairs with each end aligning at the opposite strand are thereby removed if they are too close to each other. There are two cases: (i) A read-pair with the two ends pointing towards each other ("inward-pair"), and (ii) a read-pair with the two ends pointing away from each other ("outward-pair"). Inward-pairs spanning only a short region may be caused by uncut DNA. Outward-pairs spanning only a short region can be a result of self-ligation (see Supplementary Figure 1 in [7]). Enable the filter and leave the values at default. Start processing by clicking on `map reads`. The procedure should not take too long as well: HiCat can map around 7.5 million read-pairs per minute to 823'377 *Hind*III restriction fragments of the mouse genome on our test-system [6].

The "mapped read-pair table" has eight tab-separated columns: `chromA, strandA,`

`posA, fragA, chromB, strandB, posB, fragB`, where `fragA` and `fragB` refer to the ID of a the fragment, to which the read-ends map to (the ID equals to the column `fragmentNumber` in the fragment tables (see 2.4.2)). The "reduced matrix for R" holds only the read counts per frament pair (three columns: `fragA, fragB, count`).

2.4.5 Add additional tracks to fragments

To correlate the Hi-C data to genomic and epigenetic features (e.g. gene density, histone modifications, or DNA methylation), these additional data have to be added to the genomic fragments. To annotate the genomic fragments, go to the tab "add tracks to fragments" and import the fragment table (e.g. fragments_HindIII.txt or fragments_100kb.txt, see 2.4.2) by clicking on `load fragments`. Once the fragments are loaded, you should see the four entries `fragmentNumber, chrom, start, end` on the left side ("current tracks in the file"). Any additional track successfully added will be displayed there as well. There are four different types of "tracks" which can be added:

- *genome annotation features (`ann_*`)*
  Examples are genes and transposons. Generally, these features can be very long (i.e. spanning multiple fragments). Short features like transcription binding sites or SNPs may also be added as genome annotation feature. However, for a large number of short features, it is faster to supply them as count feature. Possible formats are GFF and GTF (multiple feature types per file possible). For each feature, the number of elements per fragment is counted as follows: If the feature spans the entire fragment, a value of 1 is added. If the feature only partly overlaps (or is within) the fragment, a value of 0.5 is added.

- *count features (`sum_*`)*
  Examples may be short reads from an RNA-Seq experiment or small RNA sequencing project. BAM is the only possible format (only one feature per file, the feature will be named according to the file name). For each feature, the number of elements (e.g. short reads) per fragment is counted.

- *density features (`den_*`)*
  Examples may be short reads from a ChIP-seq experiment. BAM is the only possible format (only one feature per file, the feature will be named according to the file name). For each feature, the density per fragment is calculated as the number of bases covered by at least one element (e.g. short read) divided by the length of the fragment (times 100 to obtain %).

- *DNA methylation (`den_*`)*
  This is specifically for DNA-methylation (cytosin-methylation). The file must be a table with the tab-separated columns "chrom", "position", and "state" (plain text, without header). Position must be 0-based, and state either "m" or "u" (for methylated and unmethylated). To add multiple contexts (e.g. CG, CHG, CHH), use one file per context. Missing C's are treated as non-C characters. For each context, the DNA-methylation density per fragment is calculated as the percentage of methylated C's.

The names of the tracks added with HiCat start with one of three prefixes (`ann_`, `sum_`, `den_`). The prefixes are important for the data analysis in R (they are used to differentiate between the different data types). If custom tracks are added in another way, it is therefore important to supply the appropriate prefix.

A few tracks are supplied in the archive `At_pre-process_tutorial.zip`:

- `TAIR10.gff` holds the annotation of the *A. thaliana* reference genome. Add it under "genome annotation features (ann_)".
- `SRX006704.bam` is a seedling RNA-Seq sample [8]. Add it under "count features (sum_)"
- `SRR094098_trimmed.bam` is a control library (genomic sequencing) [9]. Add it under "density features (den_)"
- `CG_rep1.txt` contains DNA methylation data (only CG context) [10]. Add it under "DNA methylation (den_)"

Press `add all listed tracks` to add the tracks. If you are not sure if you already started it (especially if you have some other tasks running in the background), you can have a look at the "jobs currently in queue" list in the overview tab. Once the tracks are added, save and close the fragments. All the data would now be ready for the analysis in R. However, we recommend using the data specifically supplied for the R-tutorial, as there are a lot more additional tracks available (see next section).

Some other notes on the "add tracks to fragments". The fragments may be genomic bins with a fixed size or restriction fragments. The latter is preferred for one of the functions in R, which is more accurate using restriction fragments instead of large genomic bins (a function which tests a set of genomic regions for enrichment/depletion of the given genomic/epigenetic features). Restriction fragments can also be summarized into larger genomic bins directly in R. Note, however, that in this case the summarization is performed without taking the fragment length into account.

3 *HiCatR*: Analyzing Hi-C interaction profiles in R

To install `HiCatR`, download the package from github.com/MWSchmid/HiCat (the file is called `HiCatR_0.99.0.tar.gz`), open R, and type:

```
install.packages("/path/to/HiCatR_0.99.0.tar.gz", repos=NULL, type = "source")
```

The tutorial for the data analysis in R is supplied as R-Script. To obtain it, download and unpack the archive `Rscripts.zip`. We provide pre-processed data for the five samples introduced in the step-by-step tutorial (note, however, that these data were processed as described in [2]). Download and unpack the archive `At_tutorial_files.zip`. It contains tables with read counts per fragment pair ("reduced matrix for R"), tables with annotations for the genomic fragments, and some additional tables defining certain genomic regions. Open the tutorial-script ("HiCat-tutorial-arabidopsis.R") in R and follow the instructions in the comments.

The following section describes the most important functions in more detail (values in the brackets are the default values for a given argument). Note that this does not complement the tutorial-script in R. Try to follow the script first, and use the following section only as a reference for further information. To use the functions within R, it is crucial to load the organism-specific code right after loading the library:

```
library(HiCatR)
f.source.organism.specific.code("/path/to/organism-specific-code.R")
```

3.0.6 Reading data into R

A single Hi-C sample (which may comprise multiple "reduced matrices for R" created with *HiCat*) can be read into R using the function `f.load.one.sample()`. Multiple samples are loaded conveniently with `f.load.samples()`.

```
dataMatrix <- f.load.one.sample(
    dataDir = "/path/to/files",
    files = c("sampleA_run1.txt", "sampleA_run2.txt"),
    binSize = 1e6,
    repetitions = 50
)

dataMatrices <- f.load.samples(
    dataDir = "/path/to/files",
    sampleToFiles = list(
        sampleA = c("sampleA_run1.txt", "sampleA_run2.txt"),
        sampleB = c("sampleB_run1.txt", "sampleB_run2.txt", "sampleB_run3.txt")
    ),
    binSize = 1e6,
    repetitions = 50
)
```

- `dataMatrix` is a matrix with n*n entries, where n corresponds to the number of genomic fragments.
- `dataMatrices` is a list of matrices created by `f.load.one.sample()`. For a given sample, the matrix can be accessed using either `dataMatrixList[["sampleA"]]` or `dataMatrixList$sampleA`
- `binSize` specifies the size of the genomic bins to be used (see *HiCat*). To use restriction fragments instead of genomic bins with a fixed size, set `binSize = 0`.
- `repetitions [0]` sets the number of iterations for the normalization proposed by [11]. To disable the normalization, set `repetitions = 0`.

3.0.7 Normalization using linear regression

The HiC matrices can also be normalized using the approach proposed by [12].

```
normalizedDataMatrix <- f.normalize.like.hu(
    dataMatrix = dataMatrixSampleX,
    binSize = 1e6,
    annotation = annotationTable,
    lenCol = "length",
    gccCol = "gcContent",
    mapCol = "mappability",
    useNegativeBinomial = FALSE
)
```

- `dataMatrix` is a matrix with n*n entries, where n corresponds to the number of genomic fragments.
- `binSize` specifies the size of the genomic bins to be used (see *HiCat*). To use restriction fragments instead of genomic bins with a fixed size, set `binSize = 0`.
- `annotation` is a table holding genomic and epigenetic information, loaded with `f.read.annotation()` (see section 3.0.14).
- `lenCol` column with the length of the genomic fragments.
- `gccCol` column with the GC-content of the genomic fragments.
- `mapCol` column with the mappability of the genomic fragments.
- `useNegativeBinomial [FALSE]` indicates if the normalization shall be done using a negative binomial model (default is Poisson)

Note that the three parameters fragment length, GC-content, and mappability are not defined per default in the annotation tables created by *HiCat*. Examples on how to obtain them:

- fragment length can be calculated directly in R: `annotation$length = annotation$end - annotation$start`.
- GC-content can be imported as a "density" feature using *HiCat* (see section 2.4.5). Instead of using a regular DNA-methylation table, one can supply a table where the CG-positions are marked as methylated and the non-CG positions are marked as unmethylated. An example for an artificial chromosome "Chr1" starting with the sequence ACGTA:

  ```
  Chr1 0 u
  Chr1 1 m
  Chr1 2 m
  Chr1 3 u
  Chr1 4 u
  ```

- for mappability, one can align either artificial reads (from a chopped genome) or real genomic sequencing reads and import them as a "density" feature using *HiCat* (see section 2.4.5).

3.0.8 Correlation between samples

The similarity between different samples can be visualized using the function `f.Hi-C.correlation.matrix()` (figure 1).

```
f.HiC.correlation.matrix(
    dataMatrixList = dataMatrices,
    rDir = "/path/to/where/the/figure/is/stored",
    outfile = "aNameForTheFigureWithoutExtension",
    corMethod = "pearson",
    summaryFunction = median,
    useOnlyHighVar = TRUE
)
```

- `dataMatrixList` is a list of n*n matrices created by `f.load.samples()`.
- `corMethod ["pearson"]` specifies the method used to calculate the correlation between two bins i of two samples.
- `summaryFunction [median]` is used to summarize the correlations between the n pairs of bins between two samples.
- `useOnlyHighVar [TRUE]` tells if the bins with low variance should be ignored.

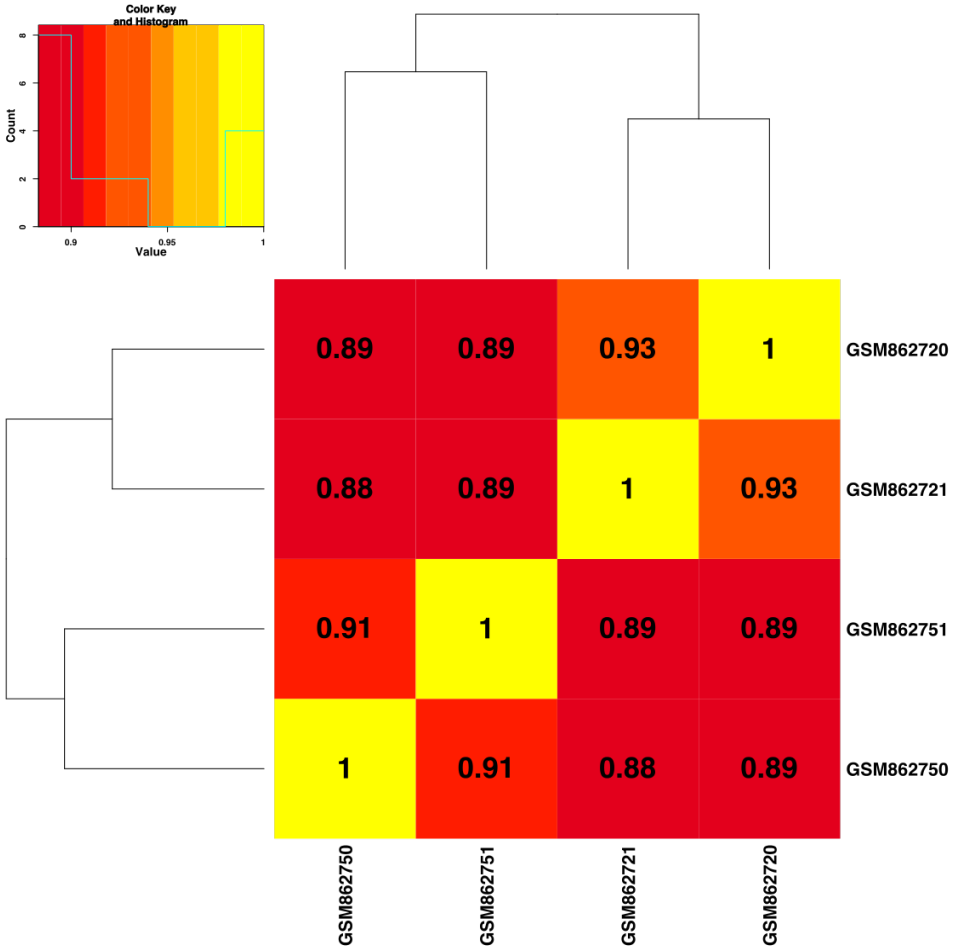

Figure 1: Correlation between four mouse Hi-C samples. The tissue types are well separated into two groups (embryonic stem cells: GSM862720, GSM862721; cortex cells: GSM862750, GSM862751 [13]; 1 mb bins).

3.0.9 Visualization of Hi-C matrices

Hi-C interaction matrices can be visualized using the function `f.plot.XY.matrix()` (figure 2 and 3).

```
f.plot.XY.matrix(
    matrixToPlot = dataMatrix,
    binSize = 1e6,
    axStep = 10e6,
    rDir = "/path/to/where/the/figure/is/stored",
    outfile = "aNameForTheFigureWithoutExtension",
    chromA = "ALL",
    startA = 0,
    endA = 0,
    chromB = "ALL",
    startB = 0,
    endB = 0,
    useLog = TRUE,
    drawGrid = FALSE,
    doNorm = FALSE,
    doCor = FALSE, # or TRUE to draw a distance-normalized, correlated Hi-C-matrix
    useSplineInterPol = TRUE
)
```

- `matrixToPlot` is a matrix created by `f.load.one.sample()`.
- `binSize` specifies the size of the genomic bins to be used (must be greater than zero, i.e. the function only takes bins with a fixed size).
- `axStep` specifies the distance between labels on the x-axis and y-axis.
- `chromA ["ALL"], startA [0], endA [0]` are genomic coordinates at the x-axis. To plot all chromosomes, set `chromA = "ALL"`.
- `chromB ["ALL"], startB [0], endB [0]` are genomic coordinates at the y-axis. To plot all chromosomes, set `chromB = "ALL"`.
- `useLog [TRUE]` tells if the data shall be transformed using `log2(data + 1)`.
- `drawGrid [FALSE]` enables a white grid that is drawn over the plot. Only useful for smaller regions.
- `doNorm [FALSE]` specifies if the data shall be distance-normalized as described in [14].
- `doCor [FALSE]` tells if the data shall be correlated before drawing (to visualize domains).
- `useSplineInterPol [TRUE]` serves to modify the color mapping.

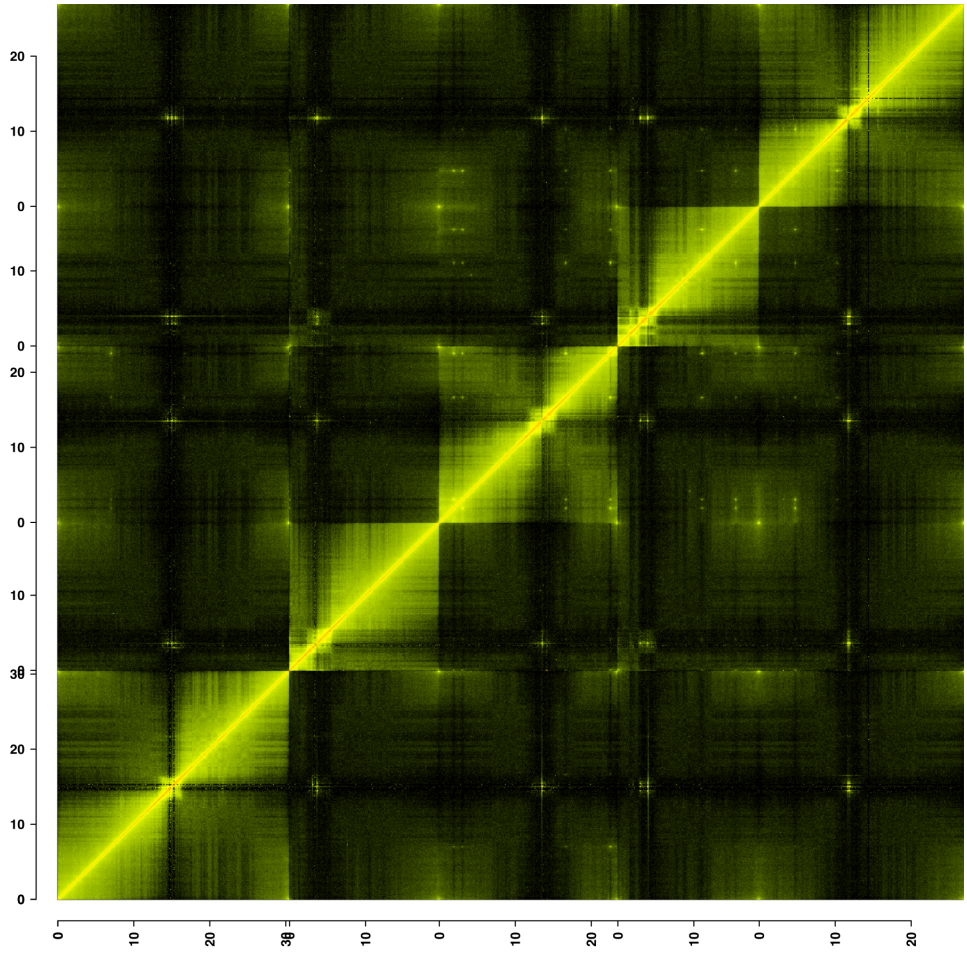

Figure 2: Visualization of Hi-C interaction frequencies in a pooled wild-type sample of *A. thaliana* [1, 2] (100 kb bins).

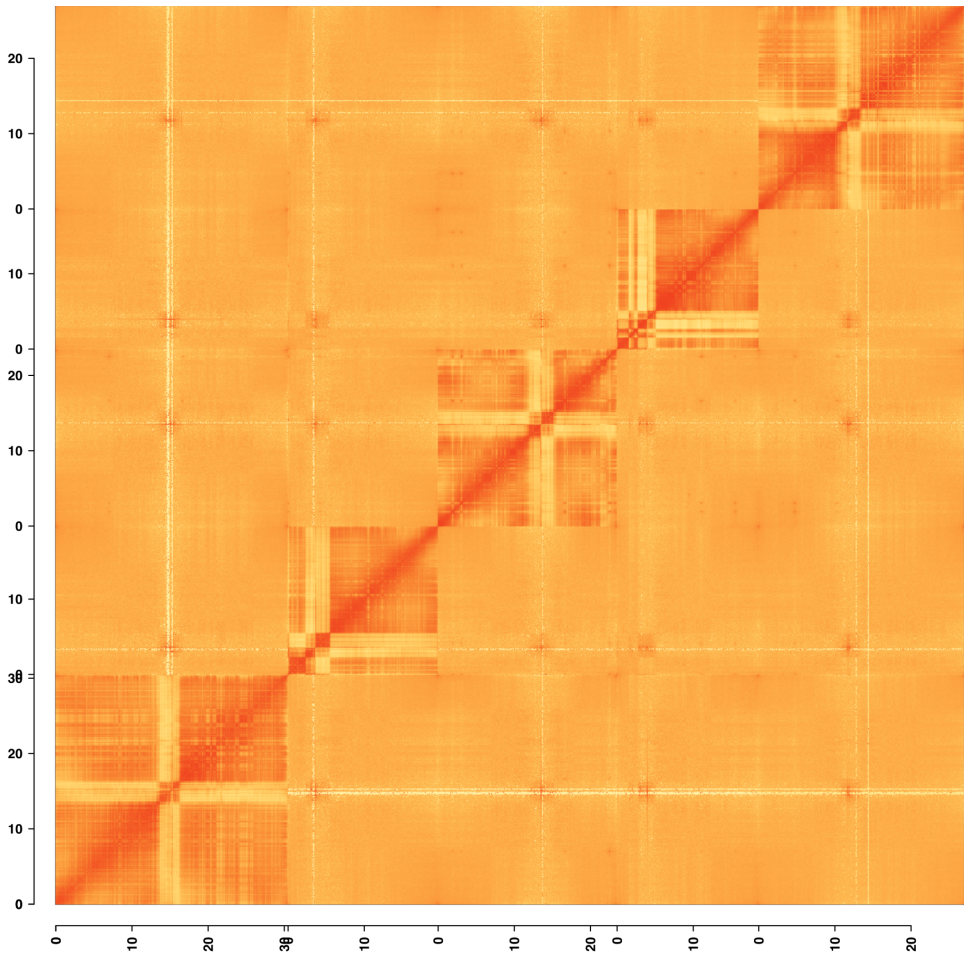

Figure 3: Visualization of distance-normalized and correlated Hi-C interaction frequencies in a pooled wild-type sample of *A. thaliana* [1, 2] (100 kb bins).

3.0.10 Comparison between Hi-C matrices - relative differences

Differences between two Hi-C samples A and B can be visualized based on the relative difference of interaction frequencies [1]. For each matrix entry (i.e. a pixel at row i and column j), the difference between the two samples is divided by the average value: $R_{ij} = (A_{ij} - B_{ij})/(A_{ij} + B_{ij})/2$. A pair of Hi-C samples can be visualized using the function `f.plot.relative.difference()`. Multiple samples are compared to each other with `f.compare.samples.relative.difference()` (figure 4).

```
f.plot.relative.difference(
    dataMatrixA = dataMatrixSampleA,
    dataMatrixB = dataMatrixSampleB,
    binSize = 1e6,
    rDir = "/path/to/where/the/figure/is/stored",
    outfile = "aNameForTheFigureWithoutExtension",
    filterZero = TRUE,
    filterThreshold = 0.95
)

f.compare.samples.relative.difference(
    dataMatrixList = dataMatrices,
    binSize = 1e6,
    rDir = "/path/to/where/the/figures/are/stored",
    outfilePrefix = "aPrefixForTheFileNames",
    filterZero = TRUE,
    filterThreshold = 0.95
)
```

- `dataMatrixA, dataMatrixB` are two matrices created by `f.load.one.sample()`.
- `dataMatrixList` is a list of n*n matrices created by `f.load.samples()`.
- `binSize` specifies the size of the genomic bins to be used (`binSize = 0` for restriction fragments).
- `outfilePrefix ["relDiff_"]` will be added in front of all file names.
- `filterZero [TRUE]` tells whether or not to filter the x percent of bins with the highest number of 0 entries.
- `filterThreshold [0.95]` specifies the fraction of bins which shall be kept if `filterZero = TRUE`.

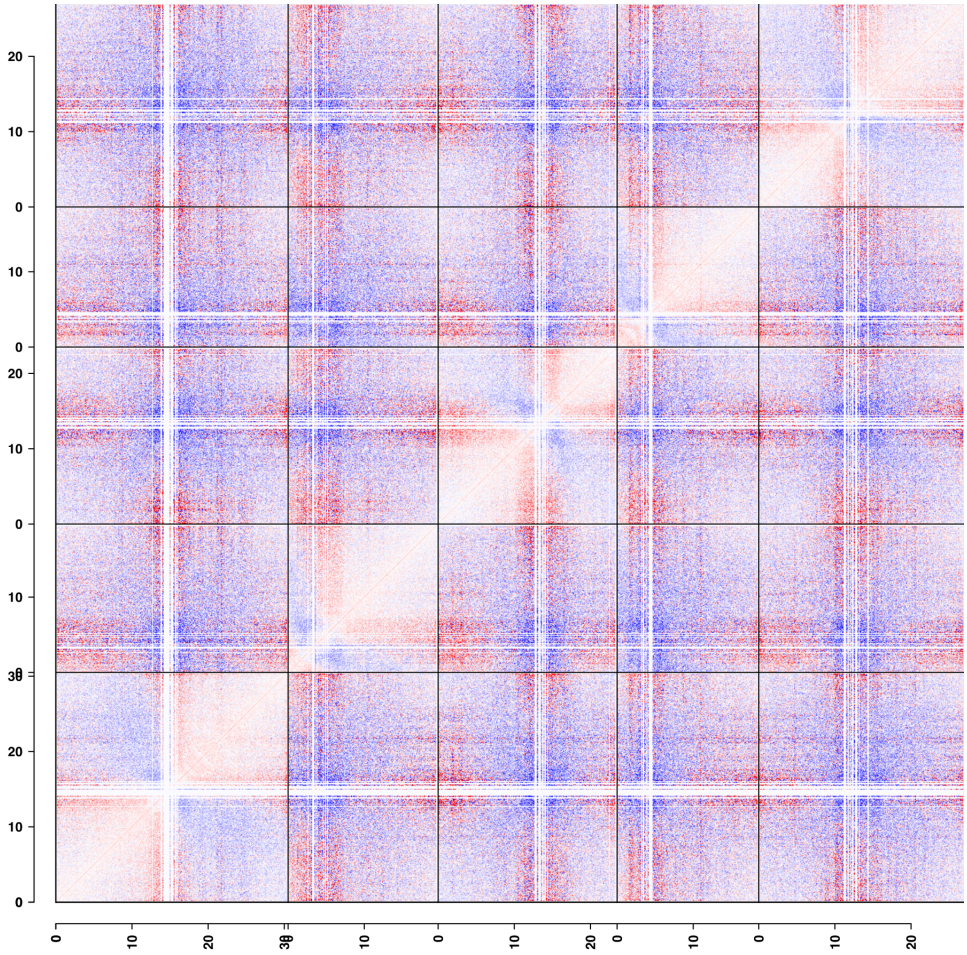

Figure 4: Enrichment (blue) and depletion (red) of interaction frequencies in the wild-type compared to the *crwn4* mutant sample of *A. thaliana* [2] (100 kb bins).

3.0.11 Comparison between Hi-C matrices - correlated differences

To see if differences are clustered, the relative differences can be correlated to each other [2]. The correlation (i.e. a pixel at row i and column j) is thereby calculated between two vectors of relative differences $C_{ij} = cor(R_i, R_j)$. A pair of Hi-C samples can be visualized using the function `f.plot.cor.difference()`. Multiple samples are compared to each other with `f.compare.samples.cor.difference()` (figure 5).

```
f.plot.cor.difference(
    dataMatrixA = dataMatrixSampleA,
    dataMatrixB = dataMatrixSampleB,
    binSize = 1e6,
    rDir = "/path/to/where/the/figure/is/stored",
    outfile = "aNameForTheFigureWithoutExtension",
    filterZero = TRUE,
    filterThreshold = 0.95
)

f.compare.samples.cor.difference(
    dataMatrixList = dataMatrices,
    binSize = 1e6,
    rDir = "/path/to/where/the/figures/are/stored",
    outfilePrefix = "aPrefixForTheFileNames",
    filterZero = TRUE,
    filterThreshold = 0.95
)
```

- `dataMatrixA, dataMatrixB` are two matrices created by `f.load.one.sample()`.
- `dataMatrixList` is a list of n*n matrices created by `f.load.samples()`.
- `binSize` specifies the size of the genomic bins to be used (`binSize = 0` for restriction fragments).
- `outfilePrefix ["corDiff_"]` will be added in front of all file names.
- `filterZero [TRUE]` tells whether or not to filter the x percent of bins with the highest number of 0 entries.
- `filterThreshold [0.95]` specifies the fraction of bins which shall be kept if `filterZero = TRUE`.

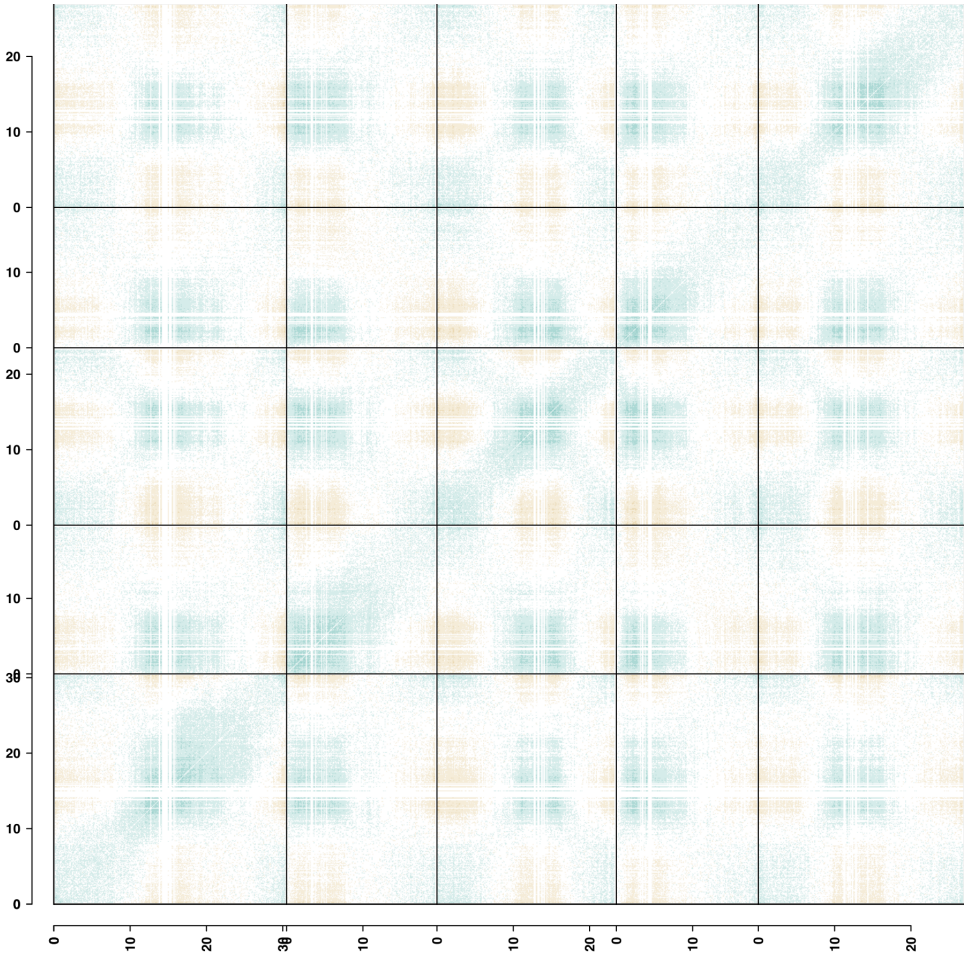

Figure 5: Correlation of differences between the wild-type and the *crwn4* mutant samples of *A. thaliana* [2] (100 kb bins).

3.0.12 Comparison between Hi-C matrices - signed differences (SDM)

An alternative way to assess the difference between two Hi-C samples are signed difference matrices (SDM) described in [2]. For each matrix entry (i.e. a pixel at row i and column j), the signed difference indicates whether a given interaction is higher or lower in sample A than sample B: $B_{ij} = sign(A_{ij} - B_{ij})$. For a pair of Hi-C samples, these signed differences can be visualized and tested for being clustered (e.g. if sample A has higher interaction frequencies than sample B over a whole chromosome arm) using the function `f.plot.signed.difference()`. Multiple samples are compared to each other with `f.compare.samples.signed.difference()` (figure 6).

```
SDMresult <- f.plot.signed.difference(
    dataMatrixA = dataMatrixSampleA,
    dataMatrixB = dataMatrixSampleB,
    binSize = 1e6,
    rDir = "/path/to/where/the/figure/is/stored",
    outfile = "aNameForTheFigureWithoutExtension",
    filterZero = TRUE,
    filterThreshold = 0.95,
    pValueThreshold = 0.01
)

SDMresultList <- f.compare.samples.signed.difference(
    dataMatrixList = dataMatrices,
    binSize = 1e6,
    rDir = "/path/to/where/the/figures/are/stored",
    outfilePrefix = "aPrefixForTheFigureNames",
    filterZero = TRUE,
    filterThreshold = 0.95,
    pValueThreshold = 0.01
)
```

- `SDMresult` is an object holding the overall P-value (`SDMresult$overallPvalue`) and the significant bin IDs (`SDMresult$significantRows`).
- `SDMresultList` is a list of objects created by `f.plot.signed.difference()`.
- `dataMatrixA, dataMatrixB` are two matrices created by `f.load.one.sample()`.
- `dataMatrixList` is a list of n*n matrices created by `f.load.samples()`.
- `binSize` specifies the size of the genomic bins to be used (`binSize = 0` for restriction fragments).
- `outfilePrefix ["relDiff_"]` will be added in front of all file names.
- `filterZero [TRUE]` tells whether or not to filter the x percent of bins with the highest number of 0 entries.
- `filterThreshold [0.95]` specifies the fraction of bins which shall be kept if `filterZero = TRUE`.
- `pValueThreshold [0.01]` specifies the significance-threshold for the individual bins.

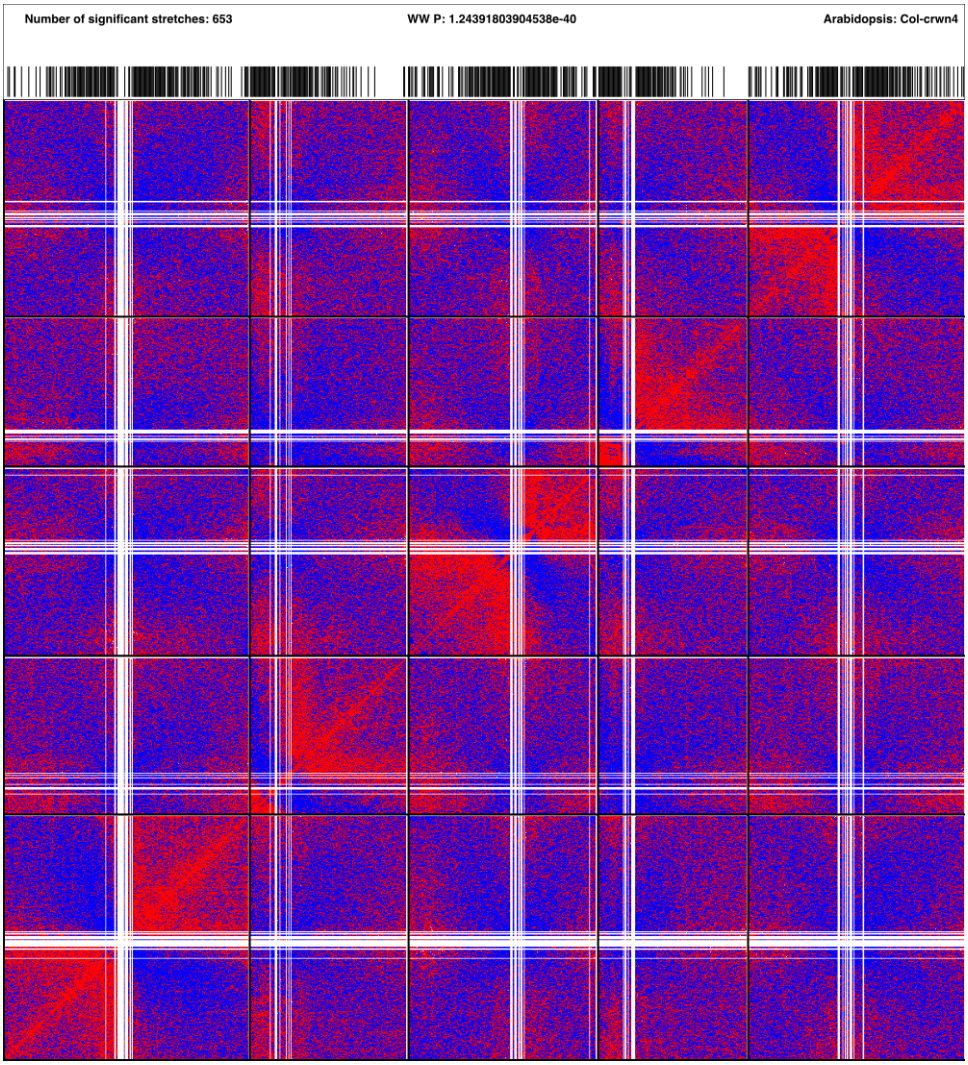

Figure 6: Visualization of the difference between the wild-type and *crwn4* mutant samples of *A. thaliana* [2] using the signed difference matrix (100 kb bins).

3.0.13 Distance-dependent decay of interaction frequencies

The distance-dependent decay of interaction frequencies (IDEs, as described in [14]) within a Hi-C sample can be calculated and visualized using the function `f.distance.decay()` (figue 7).

```
f.distance.decay(
    dataMatrix = dataMatrixSampleX,
    binSize = 1e6,
    rDir = "/path/to/where/the/figure/is/stored",
    outfile = "aNameForTheFigureWithoutExtension",
    distance = 10e6,
    regionTable = data.frame(),
    filterZero = TRUE,
    filterThreshold = 0.95
)
```

- `dataMatrix` is a matrix created by `f.load.one.sample()`.
- `binSize` specifies the size of the genomic bins to be used (must be greater than zero, i.e. the function only takes bins with a fixed size).
- `distance` specifies the distance up to which the interaction frequency decay shall be considered. Must be smaller than the smallest region (e.g. smaller than the smalles chromosome if `regionTable = data.frame()`)
- `regionTable [data.frame()]` defines specific regions in the genome which shall be assessed. Per default, each chromosome is first tested separately and a common IDE is calculated as the average between all individual IDEs. The table must have three columns (`chrom, start, end`). User defined names can be given using an optional fourth column (`name`). An example is given in the *A. thaliana* tutorial.
- `filterZero [TRUE]` tells whether or not to filter the x percent of bins with the highest number of 0 entries.
- `filterThreshold [0.95]` specifies the fraction of bins which shall be kept if `filterZero = TRUE`.

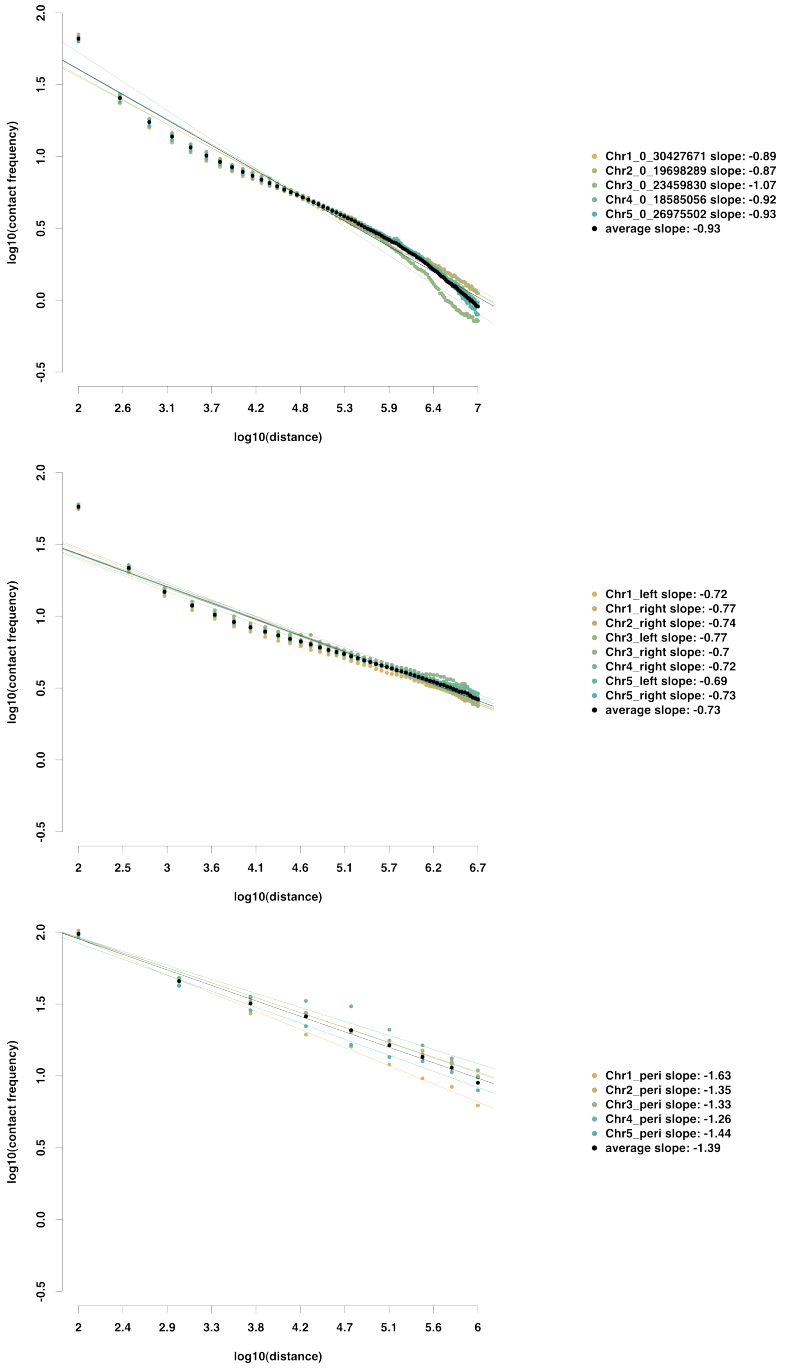

Figure 7: Distance-dependent decay of interaction frequencies along entire chromosomes (10 mb, top panel), euchromatic parts of chromosome arms (5 mb, middle panel), and heterochromatic pericentromeres (1 mb, lower panel) in a pooled wild-type sample of *A. thaliana* [1, 2] (100 kb bins).

3.0.14 Reading the annotation

Fragment annotations (additional tracks with genomic and epigenetic information) processed with *HiCat* can be read into R using the function `f.read.annotation()`. Annotations for genomic bins with fixed size can also be obtained from annotated restriction fragments with `f.read.annotation.via.fragment.annotation()`.

```
annotation <- f.read.annotation(
    annotationFile = "/path/to/file/created/with/HiCat",
    binSize = 1e6,
    useLog = TRUE
)

annotation <- f.read.annotation.via.fragment.annotation(
    annotationFile = "/path/to/file/created/with/HiCat",
    binSize = 1e6,
    useLog = TRUE
)
```

- `annotation` is a table holding genomic and epigenetic information (see *HiCat*, section 2.4.5).
- `binSize` specifies the size of the genomic bins to be used (`binSize = 0` for restriction fragments in `f.read.annotation()`).
- `useLog [TRUE]` specifies if count-like features (columns starting with `ann_`, `sum_`) shall be transformed using `log2(data + 1)`.

3.0.15 Analysis of the first principle component (PCA)

Analysis of the first principle component can be used as a tool to identify discrete structural domains [14]. The first principle component can be obtained and related to the genomic and epigenetic features (annotation) using the function `f.principle.component.analysis.and.features()` (figure 8).

```
f.principle.component.analysis.and.features(
    dataMatrix = dataMatrixSampleX,
    binSize = 1e6,
    rDir = "/path/to/where/the/results/are/stored",
    outfilePrefix = "aPrefixForTheFileNames",
    annotation = annotationTable, # or data.frame() if only PCA is requested
    regionTable = data.frame(),
    simplifiedNames = list(),
    filterZero = TRUE,
    filterThreshold = 0.95,
    pValueThreshold = 0.05,
    userLimits = c(-1, 1)
)
```

- `dataMatrix` is a matrix created by `f.load.one.sample()`.
- `binSize` specifies the size of the genomic bins to be used (must be greater than zero, i.e. the function only takes bins with a fixed size).
- `annotation [data.frame()]` is a table holding genomic and epigenetic information, loaded with `f.read.annotation()`. If no annotation is supplied, the function only performs the PCA.
- `regionTable [data.frame()]` defines specific regions in the genome which shall be analyzed individually. Per default, each chromosome is tested separately. The table must have three columns (`chrom, start, end`). User defined names can be given using an optional fourth column (`name`). An example is given in the *A. thaliana* tutorial.
- `simplifiedNames [list()]` is a list with the column names of the annotation as keys to simplified names as values (e.g. "ann_transposable_element_gene" can be replaced by "TE-gene").
- `filterZero [TRUE]` tells whether or not to filter the x percent of bins with the highest number of 0 entries.
- `filterThreshold [0.95]` specifies the fraction of bins which shall be kept if `filterZero = TRUE`.
- `pValueThreshold [0.05]` specifies the significance-threshold for the correlation and enrichment tests (only significant values are drawn in the heatmaps).
- `userLimits [c(-1, 1)]` specifies the lower and upper limit for the correlation values drawn in the heatmap.

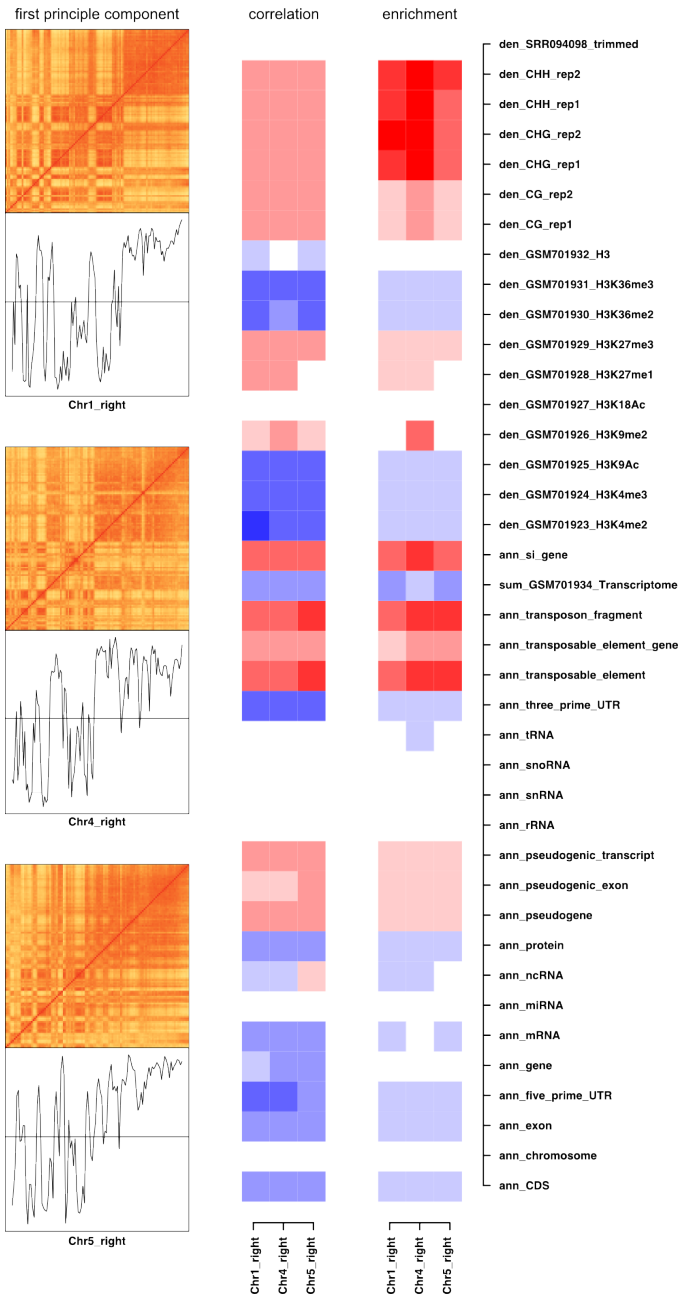

Figure 8: Left panels: Visualization of distance-normalized and correlated Hi-C interaction frequencies and the resulting first principle component. Middle: significant correlation (blue: positive, red: negative) of the first principle component with various genomic and epigenetic features. Right: significant enrichment (blue) and depletion (red) of genomic and epigenetic features in regions with positive Eigenvalues compared to regions with negative Eigenvalues. Data shown for the right arms of chromosomes 1, 4, and 5 from a pooled wild-type sample of *A. thaliana* [1, 2] (100 kb bins).

3.0.16 Test a specific set of bins for higher interaction among each other

A given set of genomic regions can be tested for preferential interaction among each other compared to sets of randomly sampled regions using the function `f.test.interaction.frequencies()`.

```
f.test.interaction.frequencies(
    dataMatrix = dataMatrixSampleX,
    binSize = 1e6,
    repetitions = 1e4,
    testRegionsTable = tableWithRegionsOfInterest,
    regionDefinitionTable = data.frame()
)
```

- `interactionResult` is a vector holding the sum of interactions between the regions of interest, the mean and standard deviation of the sampled interaction sums, and the corresponding P-value.
- `dataMatrix` is a matrix created by `f.load.one.sample()`.
- `binSize` specifies the size of the genomic bins to be used (must be greater than zero, i.e. the function only takes bins with a fixed size).
- `repetitions` specifies the number of random sets to be sampled.
- `testRegionsTable` is a table with genomic regions of interest. Columns must be `chrom, start, end`. Rownames can be freely chosen. An example is given in the *A. thaliana* tutorial.
- `regionDefinitionTable` defines specific regions in the genome from which the random sets are sampled. If a table is supplied, the regions of interest are assigned to the defined regions (unassigned will be removed) and sampling happens within the defined region (see [2] for details). If no regions are defined, the regions of interest are assigned to the whole chromosomes. Mapping of regions is done on the whole length (a test region must be entirely within a defined region to be assigned to it - unassigned test regions are removed).

3.0.17 Test a specific set of bins for enrichment/depletion of certain annotation features

A given set of genomic regions can further be tested for enrichment or depletion of certain genomic or epigenetic features using the function `f.test.regions.for.feature.enrichment.frag`

```
enrichmentResult <- f.test.regions.for.feature.enrichment.fragment.based(
    annotation = annotationTableOnRestrictionFragments,
    rDir = "/path/to/where/the/results/are/stored",
    outfilePrefix = "aPrefixForTheFileNames",
    repetitions = 1e4,
    testRegionsTable = tableWithRegionsOfInterest,
    regionDefinitionTable = data.frame(),
    simplifiedNames = list(),
    pValueThreshold = 0.05,
    countDataWasLogged = TRUE
)
```

- `enrichmentResult` is a list holding the observed values (`enrichmentResult$observed`), the enrichment (`enrichmentResult$enrichment`) compared to the random sets, and the corresponding P-values (`enrichmentResult$pValues`).

- `annotation` is a table holding genomic and epigenetic information, loaded with `f.read.annotation(..., binSize = 0)`. It is important that the annotation is based on restriction fragments and not genomic bins with fixed size (for the latter case, it is possible, but not recommended, to use `f.test.regions.for.feature.enrichment.bin.based`)

- `repetitions` specifies the number of random sets to be sampled.

- `testRegionsTable` is a table with genomic regions of interest. Columns must be `chrom, start, end`. Rownames can be freely chosen. An example is given in the *A. thaliana* tutorial.

- `regionDefinitionTable` defines specific regions in the genome from which the random sets are sampled. If a table is supplied, the regions of interest are assigned to the defined regions (unassigned will be removed) and sampling happens within the defined region (see [2] for details). If no regions are defined, the regions of interest are assigned to the whole chromosomes. Mapping of regions is done on the whole length (a test region must be entirely within a defined region to be assigned to it - unassigned test regions are removed).

- `simplifiedNames [list()]` is a list with the column names of the annotation as keys to simplified names as values (e.g. "ann_transposable_element_gene" can be replaced by "TE-gene").

- `pValueThreshold [0.05]` specifies the significance-threshold for the enrichment tests (only significant values are drawn in the heatmaps).

- `countDataWasLogged [TRUE]` tells if `useLog = TRUE` while loading the annotation with `f.load.annotation()`.

4 Building from source

If the binary is not working or you would like to implement a new feature, you can build it from the source code.

4.1 Linux

Compiling the program using QtCreator is quite easy.

- download and unpack the archive `source.zip`
- install QtCreator (on the Ubuntu repository: `qtcreator`) - make sure to use Qt 4.x.x libraries
- install zlib (on the Ubuntu repository: `zlib1g`, `zlib1g-dev`, `zlib1g-dev`)
- start QtCreator and open the `../source/HiCat.pro` file
- copy the seqan folder into one of your general include paths (e.g. `sudo cp -r seqan /usr/local/include`) or add a line `INCLUDEPATH += <path containing seqan folder>` into the *.pro file
- finally, in the QtCreater menu select `Build > Build Project ‘‘my_project’’`

4.2 Windows

Compiling on Windows requires access to VisualStudio. The following steps worked well with VisualStudio 2010 Ultimate.

- download and unpack the archive `source.zip`
- install the Qt Add-In for VisualStudio - make sure to use Qt 4.x.x libraries
- open the `../source/HiCat.pro` file (`Qt > Open Qt Project File (.pro)...`)
- download the latest zlib from zlib.net (e.g. `zlib-1.2.8.tar.gz`) and unpack the archive
- open the project properties (right-click on the project in the solution explorer) and do the following steps:
  - under `Configuration Properties > VC++ Directories`, in the field `Include Directories`, add the zlib folder
  - under `Configuration Properties > VC++ Directories`, in the field `Include Directories`, add the folder containing the seqan folder
  - under `Configuration Properties > General`, change the variable `Character Set` to `Use Multi-Byte Character Set`
  - under `Configuration Properties > C/C++ > Preprocessor` in the field `Preprocessor Definitions`, remove the `UNICODE` key-words and add the key-word `_CRT_SECURE_NO_WARNINGS`
  - under `Configuration Properties > C/C++ > Command Line`, add the options `-W2 -wd4996` in the field with the `Additional Options`
- close the project properties and compile it with `Build > Build Solution`

4.3 Mac

Currently, there are no build instructions available for the MacOS.

References

[1] G. Moissiard, S. Cokus, J. Cary, S. Feng, A. Billi, H. Stroud, D. Husmann, Y. Zhan, B. Lajoie, R. McCord, C. Hale, W. Feng, S. Michaels, A. Frand, M. Pellegrini, J. Dekker, J. Kim, S. Jacobsen, MORC family ATPases required for heterochromatin condensation and gene silencing, Science 336 (2012) 1448–1451.

[2] S. Grob, M. Schmid, U. Grossniklaus, HiC Analysis in Arabidopsis Identifies the KNOT, a Structure with Similarities to the flamenco Locus of Drosophila, Molecular Cell 55.

[3] Y. Liao, G. Smyth, W. Shi, The subread aligner: fast, accurate and scalable read mapping by seed-and-vote, Nucleic Acids Research 41 (2013) e108.

[4] H. Li, B. Handsaker, A. Wysoker, F. T., J. Ruan, N. Homer, G. Marth, G. Abecasis, R. Durbin, . G. P. D. P. Subgroup, The Sequence alignment/map (SAM) format and SAMtools, Bioinformatics 25 (2009) 2078–2079.

[5] M. Imakaev, G. Fudenberg, R. McCord, N. Naumova, A. Goloborodko, B. Lajoie, J. Dekker, L. Mirny, Iterative correction of Hi-C data reveals hallmarks of chromosome organization, Nature Methods 9 (2012) 999–1003.

[6] SYSTEM, 64 bit Kubuntu (12.04) running on an Intel® Core^{TM} i7 930@2.8 GHz with 24 Gb RAM. Input and output files were read and written from the same hard-drive (Samsung HD, 7'200 rpm).

[7] F. Jin, Y. Li, J. Dixon, S. Selvaraj, Z. Ye, A. Lee, C. Yen, A.-D. Schmitt, C. Espinoza, B. Ren, A high-resolution map of the three-dimensional chromatin interactome in human cells, Nature 503 (2013) 290–294.

[8] S. Filichkin, H. Priest, S. Givan, R. Shen, D. Bryant, S. Fox, W.-K. Wong, T. Mockler, Genome-wide mapping of alternative splicing in *Arabidopsis thaliana*, Genome Research 20 (2010) 45–58.

[9] Y. Jacob, H. Stroud, C. Leblanc, S. Feng, L. Zhuo, E. Caro, C. Hassel, C. Gutierrez, S. Michaels, S. Jacobsen, Regulation of heterochromatic DNA replication by histone H3 lysine 27 methyltransferases, Nature 466 (2010) 987–991.

[10] H. Stroud, M. Greenberg, S. Feng, Y. Bernatavichute, S. Jacobsen, Comprehensive analysis of silencing mutants reveals complex regulation of the *Arabidopsis* methylome, Cell 152 (2013) 352–364.

[11] Y. Zhang, R. McCord, Y.-J. Ho, B. Lajoie, D. Hildebrand, A. Simon, M. Becker, F. Alt, J. Dekker, Spatial Organization of the Mouse Genome and Its Role in Recurrent Chromosomal Translocations, Cell 148 (2012) 908–921.

[12] M. Hu, K. Deng, S. Selvaraj, Z. Qin, B. Ren, J. Liu, HiCNorm: removing biases in Hi-C data via Poisson regression, Bioinformatics 28 (2012) 3131–3133.

[13] J. Dixon, S. Selvaraj, Y. Feng, Y. Li, Y. Shen, M. Hu, J. Liu, B. Ren, Topological domains in mammalian genomes identified by analysis of chromatin interactions, Nature 485 (2012) 376–380.

[14] E. Lieberman-Aiden, N. Van Berkum, L. Williams, M. Imakaev, T. Ragoczy, A. Telling, I. Amit, B. Lajoie, P. Sabo, M. Dorschner, R. Sandstrom, B. Bernstein, M. Bender, M. Groudine, A. Gnirke, J. Stamatoyannopoulos, L. Mirny, E. Lander, J. Dekker, Comprehensive mapping of long-range interactions reveals folding principles of the human genome, Science 326 (2009) 289–293.