Published in final edited form as:

Curr Protoc Bioinformatics. 2019 June ; 66(1): e76. doi:10.1002/cpbi.76.

R tutorial: Detection of differentially interacting chromatin regions from multiple Hi-C datasets

John C. Stansfield¹, Duc Tran², Tin Nguyen^{2,*}, and Mikhail G. Dozmorov^{1,*}

¹Dept. of Biostatistics, Virginia Commonwealth University, Richmond, VA, 23298, USA

²Dept. of Computer Science & Engineering, University of Nevada, Reno, NV, 89557, USA

Abstract

The three-dimensional (3D) interactions of chromatin regulate cell type-specific gene expression, recombination, X chromosome inactivation, and many other genomic processes. Chromatin conformation capture (Hi-C) technologies capture the structure of the chromatin on a global scale by measuring all vs. all interactions and can provide new insights into genomic regulation. The workflow presented here describes how to analyze and interpret a comparative Hi-C experiment. We describe the process of obtaining Hi-C data from public repositories and give suggestions for pre-processing pipelines if the user intends to analyze their own raw data. We then describe the data normalization and comparative analysis process. We present the three protocols describing the usage of the multiHiCcompare, diffHic, and FIND R packages for performing a comparative analysis of Hi-C experiments. Finally, visualization of the results and downstream interpretation of the differentially interacting regions are discussed. The bulk of this tutorial uses the R programming environment and can be performed on most operating systems and a single computer.

Keywords

Hi-C; chromosome conformation capture; normalization; comparison; differential analysis; HiCcompare; multiHiCcompare

*To whom correspondence should be addressed: Mikhail G. Dozmorov, Virginia Commonwealth University, Richmond, VA, 23298, 804-827-2055, mikhail.dozmorov@vcuhealth.org; Tin Nguyen, University of Nevada, Reno, NV, 89557, 775-784-6619, tinn@unr.edu.

Internet Resources

<https://www.aidenlab.org/>

The website for the Aiden Lab - juicer software and Hi-C data sources.

<https://github.com/mirnylab/cooler>

GitHub page for cooler - Hi-C data storage and database.

<https://github.com/nservant/HiC-Pro>

GitHub page for HiC-Pro - Hi-C alignment pipeline.

<https://github.com/dozmorovlab/HiCcompare>

GitHub page for HiCcompare.

<https://github.com/dozmorovlab/multiHiCcompare>

GitHub page for multiHiCcompare.

<https://bioconductor.org/packages/devel/bioc/html/diffHic.html>

Bioconductor page for diffHic.

<https://bitbucket.org/nadhir/find>

Bitbucket page for FIND.

Introduction

Early analysis of individual Hi-C datasets illuminated basic properties of the 3D structure of the genome (A/B compartments, Topologically Associated Domains (TADs), and chromatin loops (Lieberman-Aiden et al., 2009; Imakaev et al., 2012a; Yaffe and Tanay, 2011; Dixon et al., 2012; Rao et al., 2014). One of the most important tasks in functional genomics studies is the detection of differences between two or more conditions (Dixon et al., 2015; Bonev et al., 2017; Rao et al., 2017), e.g., tumor-normal states (Taberlay et al., 2016; Rickman et al., 2012; Barutcu et al., 2015). Analogous to the differential gene expression analysis, the comparative analysis of Hi-C datasets is intended to reveal pairs of regions which are differentially interacting between conditions. These regions may be associated with loss or gain of TAD boundaries, change in TAD sizes, breaking or establishing promoter-enhancer interactions, thus pointing toward regulatory consequences. The detection of differentially interacting chromatin regions requires at least two Hi-C datasets.

Analysis of two or more Hi-C datasets poses a challenge related to a potential bias between datasets. DNA sequence-driven biases affect datasets to a similar extent because comparisons will typically be between the same species which means that they have a similar DNA sequence. In contrast, sequencing technology-driven biases unpredictably affect each dataset. Left unaccounted for, these biases may be mistaken for true biological differences. Thus, joint normalization approaches for removing the between-dataset biases are needed.

The diffHiC R package (Lun and Smyth, 2015) pioneered the removal of between-dataset biases by implementing joint MA (Minus difference-Average expression) normalization adopted from microarray technology (Dudoit et al., 2002). However, MA normalization does not account for the central property of Hi-C data - the power-law decay of chromatin interaction frequencies as the distance between interacting regions increases (Fudenberg et al., 2011; Lieberman-Aiden et al., 2009; Dekker J, 2013; Lajoie, 2015). The HiCcompare R package was the first to implement a distance-aware modification of the MA normalization, called MD (Minus difference-Distance) joint normalization (Stansfield et al., 2018b), with similar techniques being developed based on this approach (Fletez-Brant et al., 2017). The FIND method uses matrices individually normalized using Knight-Ruiz (KR) normalization (Knight and Ruiz, 2012), thus leaving the between-dataset biases unaccounted for, and exploits a spatial Poisson regression to detect differentially interacting regions (Djekidel et al., 2018). The multiHiCcompare package (Stansfield et al., 2018a) extends the MD joint normalization of two Hi-C dataset to a cyclic normalization process for joint normalization of multiple Hi-C datasets.

This unit is intended as a guide for the comparative analysis of Hi-C data to detect differentially interacting chromatin regions while accounting for the between-dataset biases. We will discuss the joint normalization of Hi-C data and the detection of statistically significant differences between (groups of) Hi-C datasets. In our workflow, we will mostly be using the R and Bioconductor environments. Several packages will be discussed including HiCcompare, multiHiCcompare, diffHic, FIND, and HiTC. The workflow has been tested on Windows 10 (Cygwin command-line interface or the Windows 10 Linux

subsystem), MacOS Sierra, and CentOS Linux distribution. Basic Protocol 1 (Fig. 1) describes the comparative analysis process using multiHiCcompare along with downstream interpretation of the results. Basic Protocol 2 discusses an alternate approach to comparative analysis of Hi-C data using diffHic. Finally, Basic Protocol 3 shows the steps for using FIND to perform a comparative analysis.

Basic Protocol 1: Performing a comparative analysis of Hi-C data using multiHiCcompare

Introduction

Here we describe the process of obtaining public Hi-C data or pre-processing the users data and the steps of a comparative analysis using multiHiCcompare. multiHiCcompare provides methods for the joint normalization of multiple Hi-C datasets and a General Linear Model (GLM) based approach for performing the differential analysis. Similar to most sequencing data, Hi-C data starts out as paired-end reads stored in fastq files. These fastq files can be very large depending on the depth of the sequencing. Several Hi-C data processing pipelines exist for the purpose of converting raw Hi-C data into text-based chromatin interaction matrices (Ay and Noble, 2015). Researchers looking to generate their Hi-C experiments will need to familiarize themselves with the Hi-C data processing pipelines to convert their raw data into chromatin interaction matrices (Lajoie, 2015). However, those who are interested in making use of the wide range of public Hi-C data deposited on Gene Expression Omnibus (GEO) repositories can normally bypass the data processing steps, as most deposited Hi-C data also includes the processed chromatin interaction matrices. These matrices are typically stored in the text-based .hic or HDF5-based .cool formats developed by Aiden lab (<http://aidenlab.org/data.html>) and Mirny lab (<http://cooler.csail.mit.edu/coolers>), respectively, and can be converted to plain text files containing chromatin interactions in sparse matrix format (only non-zero interactions are stored, described below).

As with any sequencing data, Hi-C datasets contain biases. There are two primary sources of bias, sequence- and technology-driven. The DNA sequence-driven biases include GC content, chromatin accessibility, and mappability (Yaffe and Tanay, 2011; O'Sullivan et al.), which tend to be consistent across datasets generated for the same organism. The technology-driven biases include cross-linking preferences, restriction enzyme choice, batch effects, and biotin labeling (Lun and Smyth, 2015). The technology-driven biases affect the data unpredictably and thus are harder to model. The multiHiCcompare R package was specifically designed to correct for the technology-driven biases between datasets. Once biases have been corrected, multiHiCcompare can compare the Hi-C datasets in different experimental groups for differences in chromatin interactions. For simple experiments, the Fisher's exact test can be used, and for more complex experimental designs, the GLM framework should be used. Finally, this protocol will detail several interpretation-oriented analyses that can be performed using the results of multiHiCcompare.

Necessary Resources

Hardware—A computer with internet access, at least 35GB of free hard drive space (if the user wishes to perform the example analysis), and 8GB of RAM.

Software—The R programming environment (version 3.5.0), a Unix based command-line interface (e.g., bash on Linux and MacOS, or Cygwin or the Windows 10 Linux subsystem), and a web browser.

Files—Hi-C sequencing reads in fastq format or pre-processed Hi-C matrices. Downloading and extracting the necessary files for this example protocol will be discussed in the following sections.

Obtaining Data

Public Hi-C data is available from several sources. GEO (<https://www.ncbi.nlm.nih.gov/geo/>) catalogs the data for many studies and a simple search for “Hi-C” returns 2,329 hits (as of November 6, 2018). Additionally, the Aiden Lab website (<https://www.aidenlab.org/>) lists many high-quality datasets which they have generated. Finally, there is the cooler repository (<https://github.com/mirnylab/cooler>) which provides a database of Hi-C data ready for download. More Hi-C studies and data can be found in our GitHub repository (https://github.com/mdozmorov/HiC_data).

Many sources of Hi-C data that are available for download are stored in a chromatin contact matrix text format. This can be in the form of full contact matrices (an $N \times N$ matrix) where each cell represents an interaction between two genomic regions. However, since this matrix is symmetric, the useful Hi-C information is effectively contained in the sparse upper triangular matrix. Such a sparse upper triangular matrix is stored in an $N \times 3$ matrix in which the columns represent the start position of the first interacting region, the start position of the second interacting region, and the interaction frequency (IF) for the interaction. These matrices do not contain entries for any pair of contacts with an IF of 0. The full matrix can be reconstructed from this $N \times 3$ matrix hence this format allows for large savings in storage space.

If the user is performing his own Hi-C experiments or the public data being used is not available in a chromatin interaction matrix format, then additional pre-processing steps are needed (Lajoie, 2015; Ay and Noble, 2015). These steps are briefly described below. The analysis steps for this tutorial require Hi-C data in a sparse upper triangular matrix format. The following section on aligning Hi-C data may be skipped for users starting with processed Hi-C data in the form of chromatin contact matrices.

Preprocessing raw Hi-C Data

A typical Hi-C experiment starting with raw data will begin with fastq files. fastq files should be familiar to anyone that has worked with other types of sequencing data. However, the workflow for Hi-C data differs from the typical DNA-seq process of dealing with fastq files. Hi-C libraries are normally sequenced using paired-end technology (Lajoie, 2015). As Hi-C data requires much deeper sequencing than typical DNA-seq experiments, the fastq file size can be much larger than those encountered with other sequencing techniques (approximately 20 times larger than a typical RNA-seq experiment). A typical Hi-C processing workflow includes mapping the reads, assigning fragments, filtering fragments, binning, bin level filtering, and balancing (normalization) of individual matrices (Lajoie,

2015). For the read mapping step, any standard alignment software can be used, such as Bowtie (Langmead, 2012) or the Burrows-Wheeler Aligner (Durbin, 2009). Although Hi-C data consists of paired-end reads, the reads are mapped using the single-end mode to map each read (of the pair) independently. This is because typical DNA-Seq aligners often assume that the distance between two reads in a pair fits a known distribution whereas the insert size of the Hi-C ligation product varies from several bases to hundreds of megabases. The theoretical maximum resolution that can be achieved with Hi-C sequencing is set by the restriction enzyme used to cut the DNA. However, most Hi-C datasets are not sequenced deep enough to reach this theoretical maximum, and typically one of a few fixed-size resolutions are chosen for analyzing the data. The typical resolutions used in Hi-C data analysis include, from low to high order, 1MB, 100KB, 50KB, 40KB, 20KB, 10KB, and 5KB. There are several Hi-C specific processing pipelines available for aligning raw Hi-C data. It is recommended to use one of the available Hi-C pipelines instead of attempting to perform the processing steps individually. Two of the more popular pipelines for aligning Hi-C data are juicer (Durand and Aiden, 2016) and HiC-Pro (Servant and Barillot, 2015).

Juicer—juicer <https://github.com/aidenlab/juicer/wiki> is a full pipeline that takes fastq files as input and aligns the data into .hic files which store contact map information. juicer can be run on Unix systems and uses GNU CoreUtils, Burrows-Wheeler Aligner (BWA), and Java. juicer can be run on the cloud, on a cluster, or a single computer. .hic files are a convenient and common storage method for processed Hi-C data. Contact maps can be extracted from .hic files using juicer or the command line tool straw. Below is an example of extracting a contact map from a .hic file using straw.

Installing straw: The straw tool can be compiled from C++ source <https://github.com/theaidenlab/straw>. Alternatively, download a pre-compiled system-specific binary file from <https://github.com/theaidenlab/straw/wiki/Download>. Make sure the file is located in one of the locations specified in the systems path variable. On Unix-based systems, make sure the binary has an executable attribute, `chmod +x straw`.

Run `./straw` without arguments to see a brief help on usage. See examples on how to use straw at <https://github.com/theaidenlab/straw/wiki/CPP#running>. Briefly, straw requires several inputs for the extraction of data from a .hic file:

```
<NONE/VC/VC_SQRT/KR> <hicFile(s)> <chr1>[:x1:x2] <chr2>[:y1:y2] <BP/FRAG>
<binsize>
```

The first field indicates the type of normalization to be applied. The Vanilla Coverage (VC), the square root of Vanilla Coverage (VC_SQRT), and Knight-Ruiz (KR) normalization techniques are available to be applied to the contact maps. Alternatively, the raw contact maps can be extracted using the NONE option.

The second field is the file name of the .hic file to be extracted. The following two fields are the chromosome numbers for the contact map desired, i.e., for the intrachromosomal map of chr1 the user would enter 1 1 in these fields. The next field determines if basepairs or

restriction fragment resolution files will be returned. Typically, the user will want to use the BP option. The final field specifies the resolution of the contact map.

Extracting data from .hic files: Let us assume that we downloaded and uncompressed the GSE63525_K562_combined_30.hic.gz file from GEO <https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE63525>:

```
wget
```

```
ftp://ftp.ncbi.nlm.nih.gov/geo/series/GSE63nnn/GSE63525/suppl/
GSE63525_K562_combined_30.hic.gz
```

```
gunzip GSE63525_K562_combined_30.hic.gz
```

To extract the raw matrix corresponding to chromosome 22 at 500kb resolution we would use the following command within the terminal:

```
./straw NONE GSE63525_K562_combined_30.hic 22 22 BP 500000 >
K562.chHCT116_r22.500kb.txt
```

This will extract the matrix from the .hic file and save it to the K562.chHCT116_r22.500kb.txt text file, in the sparse upper triangular matrix format. Typically, chromosome-specific matrices are saved in separate files.

HiC-Pro—HiC-Pro is another Hi-C processing pipeline that takes fastq files as input. HiC-Pro can be run on a single computer or a cluster. HiC-Pro can automatically apply Iterative Correction and Eigenvector decomposition (ICE) normalization to the Hi-C data in addition to producing the un-normalized contact maps. HiC-Pro's output includes two main file types, .matrix and .bed files. The .matrix files are plain text 3 column sparse upper triangular matrices with the columns bin_i , bin_j and $counts_{ij}$. The .bed file contains the genomic coordinates corresponding to each of these bin_i and bin_j .

HiC-Pro can be installed from GitHub here: <https://github.com/nservant/HiC-Pro>. It requires bowtie2 (>2.2.2), Python (>2.7), R (>3.4), g++ compiler (>4.4.0), samtools (>1.1), and Unix “sort” command. HiC-Pro was built for Unix systems; however, it also has support for Linux, Windows, and Mac through a Singularity image. HiC-Pro can be used on a cluster or a single computer.

Working with processed Hi-C data—The HiCcompare R package was designed for working with processed Hi-C data. It contains several functions which may be useful for an analysis. Hi-C data extracted from .hic files using straw can be simply read into R in the standard fashion for loading any text file containing data. HiCcompare can then be used to convert Hi-C data from sparse upper triangular matrix format into a full contact matrix using the sparse2full function. This process can be reversed using the full2sparse function. Data aligned by HiC-Pro can be converted into a more usable BEDPE format using the hicpro2bedpe function. The hicpro2bedpe function takes the .matrix and .bed files produced

by HiC-Pro as input and produces a sparse upper triangular matrix containing start and end coordinates for each interacting region.

Comparative analysis of multiple Hi-C datasets using multiHiCcompare

HiCcompare—The original HiCcompare R package can be used when only two Hi-C datasets are available to be compared (Stansfield et al., 2018b). HiCcompare provides a method for the joint normalization and difference detection of two Hi-C datasets, but cannot be generalized to higher numbers of datasets. multiHiCcompare will need to be used if more than two Hi-C datasets are to be compared (Stansfield et al., 2018a).

Obtaining and preparing the data—We begin our tutorial illustrating the processing of Hi-C datasets to prepare them for comparative analysis. First, we will need to download an example set of Hi-C data. We will use data from Rao 2017 (Rao et al., 2017). For simplicity, we will only use two replicates for each experimental condition. The experimental conditions are normal HCT-116 cells and HCT-116 cells treated with auxin for six hours. To download the .hic files from GEO run the following commands in the terminal. Note: downloading the data for this protocol will require about 30GB of hard drive space.

```
wget ftp://ftp.ncbi.nlm.nih.gov/geo/samples/GSM2795nnn/GSM2795535/suppl/GSM2795535_Rao-2017-HIC001_30.hic.gz
wget ftp://ftp.ncbi.nlm.nih.gov/geo/samples/GSM2795nnn/GSM2795536/suppl/GSM2795536_Rao-2017-HIC002_30.hic.gz
wget ftp://ftp.ncbi.nlm.nih.gov/geo/samples/GSM2809nnn/GSM2809539/suppl/GSM2809539_Rao-2017-HIC008_30.hic.gz
wget ftp://ftp.ncbi.nlm.nih.gov/geo/samples/GSM2809nnn/GSM2809540/suppl/GSM2809540_Rao-2017-HIC009_30.hic.gz
```

The next step is to extract the data from the .hic files. Now we will extract the un-normalized data from the .hic files at 100KB resolution:

```
# Unzip .hic files
gunzip *.gz

# Make directories for the contact map files
mkdir HIC001
mkdir HIC002
mkdir HIC008
mkdir HIC009

# Extract contact maps using straw by running the following commands in the terminal
# Or, put the commands into a script file, e.g., 'straw.sh', and run it
for i in {1..22}
do
    ./straw NONE GSM2795535_Rao-2017-HIC001_30.hic $i $i BP 100000 >
HIC001/HIC001.NONE.chr$i.100000.txt
done
./straw NONE GSM2795535_Rao-2017-HIC001_30.hic X X BP 100000 >
HIC001/HIC001.NONE.chrX.100000.txt
```

```

for i in {1..22}
do
    ./straw NONE GSM2795536_Rao-2017-HIC002_30.hic $i $i BP 100000 >
HIC002/HIC002.NONE.chr$i.100000.txt
done
./straw NONE GSM2795536_Rao-2017-HIC002_30.hic X X BP 100000 >
HIC002/HIC002.NONE.chrX.100000.txt

for i in {1..22}
do
    ./straw NONE GSM2809539_Rao-2017-HIC008_30.hic $i $i BP 100000 >
HIC008/HIC008.NONE.chr$i.100000.txt
done
./straw NONE GSM2809539_Rao-2017-HIC008_30.hic X X BP 100000 >
HIC008/HIC008.NONE.chrX.100000.txt

for i in {1..22}
do
    ./straw NONE GSM2809540_Rao-2017-HIC009_30.hic $i $i BP 100000 >
HIC009/HIC009.NONE.chr$i.100000.txt
done
./straw NONE GSM2809540_Rao-2017-HIC009_30.hic X X BP 100000 >
HIC009/HIC009.NONE.chrX.100000.txt

```

These steps will create four folders containing the sparse upper triangular matrices for chromosomes 1-22, and X for each sample. HIC001 and HIC002 are the two replicates for the normal HCT-116 cells and HIC008 and HIC009 are the two replicates for the auxin-treated HCT-116 cells.

We now need to read the data into R. Open R and make sure the working directory is set to the directory where the Hi-C data is stored and execute the following commands:

```

# Install, if necessary, and load necessary libraries and set up R session
if (!requireNamespace("BiocManager", quietly = TRUE)) install.packages("BiocManager")
library(readr) # install.packages("readr")
library(data.table) # install.packages("data.table")
library(dplyr) # install.packages("dplyr")
library(edgeR) # BiocManager::install("edgeR")
library(BiocParallel) # BiocManager::install("BiocParallel")
library(HiCcompare) # BiocManager::install("HiCcompare"), or, for the latest version,
devtools::install_github('dozmorovlab/HiCcompare', build_vignettes = TRUE, force = TRUE)
# install.packages("devtools")
library(multiHiCcompare) # BiocManager::install("multiHiCcompare", version = "devel")
options(scipen = 10) # Output fixed numbers, not scientific notation

# Set up parameters for reading in data
chr <- paste0('chr', c(1:22, 'X')) # Chromosome names
samples <- paste0('HIC00', c(1, 2, 8, 9)) # Sample names
res <- 10000 # Data resolution

# Read data
sample_list <- list()
chr_list <- list()
for( j in 1:length(samples)) {
  for( i in 1:length(chr)) {
    chr_list[[i]] <- read_tsv(paste0(samples[j], "/", samples[j],
                                     ".NONE.", chr[i], ".", res, ".txt"),

```



```

                                col_names = FALSE) %>% as.data.table()
  # Add column indicating the chromosome
  chr_list[[i]] <- cbind(i, chr_list[[i]])
  colnames(chr_list[[i]]) <- c('chr', 'region1', 'region2', 'IF')
}
sample_list[[j]] <- chr_list
chr_list <- list()
}

# Collapse separate chromosome lists into one table per sample
sample_list <- lapply(sample_list, rbindlist)

```

We now have a list with each entry containing the sparse upper triangular matrix for one of the Hi-C datasets:

```
sample_list[[1]]
```

	chr	region1	region2	IF
1:	1	0	0	16
2:	1	0	100000	1
3:	1	500000	500000	13
4:	1	600000	600000	1
5:	1	500000	700000	4

15004441:	23	154800000	155200000	53
15004442:	23	154900000	155200000	88
15004443:	23	155000000	155200000	138
15004444:	23	155100000	155200000	402
15004445:	23	155200000	155200000	814

The first column indicates the chromosome number, the second column is the start location in base pairs for the first interacting region, the third column is the start location for the second interacting region, and the fourth column is the interaction frequency (IF) for the interacting pair.

Joint normalization of Hi-C datasets—First, we need to create a Hicexp object using the Hi-C data:

```

# Create a Hicexp object for use by multiHiCcompare (~10 min)
# Four objects are assigned into two groups
rao2017 <- make_hicexp(data_list = sample_list, groups = c(1, 1, 2, 2))

rao2017 # class(rao2017)

Hi-C Experiment Object
2 experimental groups
Group 1 has 2 samples
Group 2 has 2 samples

```

The Hicexp object stores the Hi-C experiment data and is the main input into the other functions included in multiHiCcompare. The user can view the IF information by using the hic_table accessor function:

```
hic_table(rao2017)
```

	chr	region1	region2	D	IF1	IF2	IF3	IF4
1:	1	0	0	0	16	10	5	13
2:	1	500000	500000	0	13	19	7	4
3:	1	500000	800000	3	8	15	3	2
4:	1	700000	700000	0	668	968	260	382
5:	1	700000	800000	1	356	449	156	179

5948969:	23	155000000	155100000	1	460	425	306	323
5948970:	23	155000000	155200000	2	138	167	65	80
5948971:	23	155100000	155100000	0	2116	2391	1220	1427
5948972:	23	155100000	155200000	1	402	436	202	222
5948973:	23	155200000	155200000	0	814	1023	392	519

When comparing multiple Hi-C datasets, a joint normalization procedure increases power and reduces the number of false positives (Stansfield et al., 2018b). The multiHiCcompare R package includes two methods for the joint normalization of Hi-C data, cyclic loess and fast loess (fastlo) (Ballman, 2004). We will normalize the data using fastlo.

```
# MD plots before normalization
MD_hicexp(rao2017, plot.chr = 1, plot.loess = TRUE)

# Normalize (~2 min)
rao2017 <- fastlo(rao2017)

# Plot normalization results
MD_hicexp(rao2017, plot.chr = 1, plot.loess = TRUE)

# Print normalized IFs
pander::pandoc.table(head(hic_table(rao2017)))
```

chr	region1	region2	D	IF1	IF2	IF3	IF4
1	0	0	0	14.2	5.515	8.543	15.62
1	500000	500000	0	11.53	10.75	11.81	4.94
1	500000	800000	3	4.89	8.174	5.851	3.668
1	700000	700000	0	594.3	966.6	275.3	406.2
1	700000	800000	1	252.3	326.8	224.2	241.8
1	700000	900000	2	85.74	132.6	49.21	73.79

The IFs in the `hic_table` slot have been updated with their normalized values. The MD plots (Fig. 2) show that the normalization has been performed correctly, and the cloud of points is centered and symmetric around 0 indicating that any biases between datasets have been removed. The MD plot displays unit genomic distance on the x-axis and the log2 difference between the two datasets on the y-axis. Any shift of the points away from $y = 0$ represents scaling differences between the datasets. The loess fit to the data on the MD plot will also model any trend biases between the datasets. Correctly normalized data should be centered around $y = 0$ and symmetric (without any clear trends) on the MD plot.

Note that if multiple cores are available, the runtime of `multiHiCcompare` can be sped up by using the parallel option. `multiHiCcompare` was built with the Bioconductor `BiocParallel` package. The number of cores to be used in parallel processing can be set as follows:

```
library(BiocParallel) # BiocManager::install("BiocParallel")
# Check how many cores are available
numCores <- parallel::detectCores()

# Set the number of cores at least one less than the total number
if(Sys.info()['sysname'] == "Windows") {
  # Windows settings
  register(SnowParam(workers = numCores - 1), default = TRUE)
} else {
  # Unix settings

  register(MulticoreParam(workers = numCores - 1), default = TRUE)
}
```

Now that multiple cores are registered, the user can utilize parallel processing in any of the normalization and difference detection steps by setting `parallel = TRUE` in the function options.

Difference detection—Now that we have jointly normalized our data, we are ready to compare the conditions to find differentially interacting chromatin regions. For this example, we only have two conditions and no other covariates. Thus, we can use the exact test for our comparison:

```
# Perform exact test (~10 min)
# May use "parallel = TRUE" option to speed up computations
rao2017 <- hic_exactTest(rao2017, parallel = TRUE)

# Plot a composite MD plot with the results of a comparison
MD_composite(rao2017, plot.chr = 1)

# Print results as a data frame
pander::pandoc.table(head(results(rao2017)))
```

chr	region1	region2	D	logFC	logCPM	p.value	p.adj
1	0	0	0	0.4226	0.7776	0.6929	0.8819
1	500000	500000	0	-0.5822	0.6359	0.5607	0.8138
1	500000	800000	3	-0.6444	2.419	0.6813	0.7784
1	700000	700000	0	-1.72	6.216	0.00002573	0.003778
1	700000	800000	1	-0.4498	6.471	0.02507	0.05412
1	700000	900000	2	-1.19	4.879	0.0002512	0.001082

```
# Save the Hicexp object
save(rao2017, file = 'rao2017.RDA')

# To start the downstream analysis
# without re-running multiHiCcompare load the saved file
# Load('rao2017.RDA')
```

Here we can see the results. The composite MD plot highlights where the significantly different interactions are occurring in relation to distance and the fold change of the difference between groups (Fig. 3). The results table shares the same first four columns with the `hic_table`, but the following columns indicate the results of the exact test. `logFC` is the log fold change difference between the experimental groups, `logCPM` is the log counts per million between the samples, `p.value` is the un-adjusted p-value for the exact test, and `p.adj` is the false discovery rate (FDR) corrected p-value from the exact test.

Alternate GLM example—For more complex experiments the exact test is no longer sufficient, and the general linear model (GLM) framework must be used. If for example, we have some other covariate of interest that we wish to control for or if there are more than two experimental groups the GLM functionality of `multiHiCcompare` should be used. Here we show an example GLM analysis using two additional replicates from Rao 2017 which come from different biological samples. First, we need to download the additional files:

```
# Download additional two samples
wget ftp://ftp.ncbi.nlm.nih.gov/geo/samples/GSM2795nnn/GSM2795538/suppl/GSM2795538_Rao-2017-HIC004_30.hic.gz
wget ftp://ftp.ncbi.nlm.nih.gov/geo/samples/GSM2809nnn/GSM2809543/suppl/GSM2809543_Rao-2017-HIC012_30.hic.gz

# Unzip .hic files
gunzip *.gz

# Make directories for the contact map files
mkdir HIC004
mkdir HIC012

# Extract contact maps using straw by running the following commands in the terminal
# Or, putting the commands into a script file, e.g., 'straw.sh', and running it
for i in {1..22}
do
  ./straw NONE GSM2795538_Rao-2017-HIC004_30.hic $i $i BP 100000 >
HIC004/HIC004.NONE.chr$i.100000.txt
done
./straw NONE GSM2795538_Rao-2017-HIC004_30.hic X X BP 100000 >
HIC004/HIC004.NONE.chrX.100000.txt

for i in {1..22}
do
  ./straw NONE GSM2809543_Rao-2017-HIC012_30.hic $i $i BP 100000 >
HIC012/HIC012.NONE.chr$i.100000.txt
done
./straw NONE GSM2809543_Rao-2017-HIC012_30.hic X X BP 100000 >
HIC012/HIC012.NONE.chrX.100000.txt
```

Then read the data into R and create a Hicexp object as before:

```
# Set up parameters for reading in data
chr <- paste0('chr', c(1:22, 'X')) # Chromosome names
samples <- paste0('HIC0', c('01', '02', '04', '08', '09', '12')) # Sample names
res <- 100000 # Data resolution

# Read data
sample_list <- list()
chr_list <- list()
for( j in 1:length(samples)) {
  for( i in 1:length(chr)) {
    chr_list[[i]] <- read_tsv(paste0(samples[j], "/", samples[j],
                                     ".NONE.", chr[i], ".", res, ".txt"),
                             col_names = FALSE) %>% as.data.table()

    # Add column indicating the chromosome
    chr_list[[i]] <- cbind(i, chr_list[[i]])
    colnames(chr_list[[i]]) <- c('chr', 'region1', 'region2', 'IF')
  }
  sample_list[[j]] <- chr_list
  chr_list <- list()
}

# Collapse separate chromosome lists into one table per sample
sample_list <- lapply(sample_list, rbindlist)

# Create a Hicexp object for use by multiHiCcompare
# Add the covariate data.frame for biological sample source
rao_glm <- make_hicexp(data_list = sample_list, groups = c(1, 1, 1, 2, 2, 2),
                      covariates = data.frame(biosample = c(1,1,2,1,1,2)))
```

Now we can normalize as was done before:

```
rao_glm <- fastlo(rao_glm, parallel = TRUE)
```

Now we are ready to use the GLM functionality of multiHiCcompare:

```
# View covariates
meta(rao_glm)
# Perform GLM
# Make design matrix
d <- model.matrix(~factor(meta(rao_glm)$group) + factor(meta(rao_glm)$biosample))
# Plug into GLM function
rao_glm <- hic_glm(rao_glm, design = d, coef = 2)

# Plot a composite MD plot with the results of a comparison
MD_composite(rao_glm, plot.chr = 1, D.range = 0.2)

# Print results as a data frame
results(rao_glm)
```

The results of the above GLM analysis are now controlled for biological sample source. The resulting Hicexp object can again be visualized the same as done above in the exact test case example.

Downstream analysis and interpretation of differentially interacting regions

The identification of differentially interacting chromatin regions (DIRs) opens up a problem of interpretation - what is so special about these regions from a genome regulation perspective? Answers to the following questions may help to better understand the regulatory role of differentially interacting regions.

- **Visualization of DIRs.** A Manhattan-like plot of DIRs may inform us about abnormalities or reveal chromosome-specific enrichment of differentially interacting regions.
- **Overlap between differentially expressed genes and DIRs.** If gene expression measurements are available, a list of differentially expressed genes may be tested for overlap with DIRs. The goal of this analysis is to establish a formal link between DIRs and changed gene expression.
- **Functional enrichment of genes overlapping DIRs.** DIRs may disrupt the regulation of genes overlapping them. The goal of this analysis is to test whether genes overlapping DIRs are enriched in a canonical pathway or share a common function.
- **Overlap enrichment between TAD boundaries and DIRs.** DIRs may correspond to TAD boundaries that are deleted or created. Thus, it is important to test DIRs for significant overlap with TAD boundaries detected in either condition or only in boundaries changed between the conditions. Similar overlap enrichment can be calculated between DIRs and any genomic annotation.
- **Overlap between DIRs and binding sites.** DIRs may correspond to locations where proteins bind with the DNA such as CTCF sites. Thus, it may be of interest to check for overlap between binding site locations and DIRs.

Visualizing differentially interacting chromatin regions (DIRs)—Regions which are frequently detected as differentially interacting may be visualized by using the manhattan plot-like plotting function provided by multiHiCcompare. The function `manhattan_hicexp` allows the user to make a Manhattan plot showing the regions that are either detected as significantly interacting with any other regions (summarized p-value) or

frequently detected as significantly interacting (number of times a region is significantly differentially interacting with other regions). The p-value summarization options include the addCLT (default) (Nguyen, 2016), fisher (Fisher, 1950), and stouffer (Stouffer, 1949) methods to combine the p-values for each region to produce a plot of the most significant regions. The count method creates a plot where the height corresponds to the number of times a region was detected as significant. The goal of these plots is to visualize the most significantly differentially interacting regions in the context of the linear genome.

The manhattan_hicexp plots summarize p-values (method = 'addCLT', fisher, or stouffer) on the $-\log_{10}(pvalue)$ Y-axis (Fig. 4A), or the number of times a region was detected as significant (method = 'count', Fig. 4B). Statistics for *all* regions are plotted. The higher the dots are, the more significant/more frequent a region was detected as significantly differentially interacting. Use plot.chr to focus on any given chromosome:

```
manhattan_hicexp(rao2017, method = 'addCLT')

manhattan_hicexp(rao2017, method = 'count', plot.chr = 18)
```

It may be of interest to take a more in-depth look at the most significant regions that were detected as differentially interacting many times. We can get started with this by using the topDirs function that gives us a data.frame of the regions and the count for the number of times each region was detected as differentially interacting, along with the Fisher combined p-value of the detected interactions. The topDirs function is an analog of the limma::topTable and edger::topTags functions in that it allows us to filter the results by the average log fold-change (logfc_cutoff), the average interaction frequency (the higher the average frequency the more confident we are in the detected difference, logcpm_cutoff), the adjusted p-value cutoff (p.adj_cutoff), and the distance cutoff (D_cutoff). The topDirs function allows us to focus on the most significant regions while filtering out less interesting regions.

The return_df = 'bed' option gives us a summary of the regions which are found to be interacting at least one time or more:

```
counts <- topDirs(rao2017, logfc_cutoff = 1, logcpm_cutoff = 2,
                  p.adj_cutoff = 0.01, return_df = 'bed')
pander::pandoc.table(head(counts))
```

chr	start	end	count	avgD	avgLogFC	avgLogCPM	avgP.adj
chr1	121400000	121499999	61	63.28	1.097	9.15	2.7462E-41
chr16	8600000	8699999	56	24.5	-1.589	7.667	3.8393E-37
chr23	114900000	114999999	55	26.78	-2.107	7.234	5.0491E-37
chr16	58800000	58899999	51	30.2	-1.473	8.22	3.8325E-34
chr16	89800000	89899999	48	51.62	1.601	8.451	4.0538E-32
chr17	26800000	26899999	47	65.34	-2.057	7.425	1.6855E-31

We can now use the counts data.frame as an input for plotting the p-values of the top DIRs, summarized by Fisher's method (Fig. 5A):

```
plot_pvals(counts)
```

We can also plot the counts (Fig. 5B):

```
plot_counts(counts)
```

To zoom in on a particular chromosome the plot.chr option can be used:

```
plot_counts(counts, plot.chr = 2)
```

The return_df = 'pairedbed' will give the results in the form of interacting pairs:

```
pairs <- topDirs(rao2017, logfc_cutoff = 2, logcpm_cutoff = 4,
                 p.adj_cutoff = 0.01, return_df = 'pairedbed')
pander::pandoc.table(head(pairs))
```

chr1	start1	end1	chr2	start2	end2	D	logFC	logCPM	p.value	p.adj
chr1	2400000	2499999	chr1	2600000	2699999	2	-2.032	4.95	1.9161E-10	3.8235E-09
chr1	2600000	2699999	chr1	6400000	6499999	38	-3.673	5.063	1.2405E-06	1.9435E-03
chr1	2600000	2699999	chr1	6500000	6599999	39	-3.535	5.016	1.1798E-06	1.9435E-03
chr1	3800000	3899999	chr1	4500000	4599999	7	-2.548	6.334	9.8812E-19	3.4347E-16
chr1	3800000	3899999	chr1	4600000	4699999	8	-2.095	6.404	3.7784E-14	6.8404E-12
chr1	3800000	3899999	chr1	4700000	4799999	9	-2.005	6.295	1.3393E-12	1.9727E-10

The coordinates of differentially interacting regions may be saved as .bed files for downstream analysis in tools such as GenomeRunner (Dozmorov, 2016), LOLA (Sheffield and Bock, 2016), or visualization in, e.g., UCSC Genome Browser (Karolchik and Kent., 2009):

```
# Regular BED format
write_tsv(counts[, c('chr', 'start', 'end', 'count')],
          path = 'detected_regions.bed', col_names = FALSE)
# Paired BED format
write_tsv(pairs, path = 'detected_regions.pairedbed',
          col_names = FALSE)
```

Sometimes, a BED file of all regions in the genome needs to be saved, to be used as a “background” for random sampling:

```
# Get List of all 100KB regions in genome
regions <- topDir(rao2017, logfc_cutoff = 0, logcpm_cutoff = -1,
                 D_cutoff = 0, p.adj_cutoff = 1, alpha = 2,
                 return_df = 'bed' )

# Order regions
regions <- regions[order(chr, start, end), ]
# Remove unnecessary columns
regions <- regions[, c('chr', 'start', 'end')]
# Write into BED format
write_tsv(regions, path = 'all_regions.bed', col_names = FALSE)
```

Overlap between differentially expressed genes and DIRs—If gene expression data is available, DIRs may be checked for statistically significant overlap with differentially expressed (DE) genes. The hypothesis here is that regions detected as differentially interacting harbor genes that change their gene expression due to changes in chromatin interactions. In our example, we obtain a list of genes differentially expressed between the normal cells and auxin treated cells (Rao et al., 2017) and test whether they are co-localized with DIRs. First, we get the genomic coordinates (hg19/GRCh37) of all differentially interacting regions:

```
library(GenomicRanges) # BiocManager::install("GenomicRanges")
# Make GRanges from significant regions
sig.regions <- topDir(rao2017, logfc_cutoff = 1, p.adj_cutoff = 10^-15,
                    return_df = 'bed')
sig.regions.gr <- makeGRangesFromDataFrame(sig.regions,
                                           seqnames.field = 'chr',
                                           start.field = 'start',
                                           end.field = 'end',
                                           keep.extra.columns = TRUE)
```

Next, we get the genomic coordinates of all protein-coding genes in the genome. They will be used for a permutation test, to assess the average probability of overlap between DIRs and genes:

```
# Install annotables package for gene locations
# devtools::install_github("stephenturner/annotables")
library(annotables)
library(dplyr)

# Use annotables for hg19 symbols
hg19_symbols <- grch37 %>% # Get genomic coordinates for hg19/GRCh37 genome assembly
  subset(., biotype == "protein_coding") %>% # For protein-coding genes only
  subset(., chr %in% c(1:22, 'X')) # On autosomes and X chromosome
# Make X chromosome numeric for compatibility with Hi-C data conventions
hg19_symbols$chr[hg19_symbols$chr == 'X'] <- 23
hg19_symbols$chr <- paste0('chr', hg19_symbols$chr)
hg19_symbols$strand <- ifelse(hg19_symbols$strand == -1, '-', '+')
head(hg19_symbols)

# A tibble: 6 x 9
  ensgene entrez symbol chr start end strand biotype description
  <chr>    <int> <chr>   <chr> <int> <int> <chr>   <chr>   <chr>
1 ENSG000~ 7105 TSPAN6 chr23 9.99e7 9.99e7 - protei~ tetraspanin ~
2 ENSG000~ 64102 TNMD chr23 9.98e7 9.99e7 + protei~ tenomodulin ~
3 ENSG000~ 8813 DPM1 chr20 4.96e7 4.96e7 - protei~ dolichyl-pho~
4 ENSG000~ 57147 SCYL3 chr1 1.70e8 1.70e8 - protei~ SCY1-like 3 ~
5 ENSG000~ 55732 C1orf1~ chr1 1.70e8 1.70e8 + protei~ chromosome 1~
6 ENSG000~ 2268 FGR chr1 2.79e7 2.80e7 - protei~ feline Gardn~
```

Then, we need to download the list of differentially expressed genes from GEO:

```
wget ftp://ftp.ncbi.nlm.nih.gov/geo/series/GSE106nnn/GSE106886/suppl/
GSE106886_Rao-2017-RAD21notreat_vs_RAD21treat.Genes.DESeq2.txt.gz
```

```
gunzip GSE106886_Rao-2017-RAD21notreat_vs_RAD21treat.Genes.DESeq2.txt.gz
```

Now, we read the list of differentially expressed genes into R, get their genomic coordinates (hg19 genome assembly), and create the GRanges object:

```
de.genes <- read.table('GSE106886_Rao-2017-RAD21notreat_vs_RAD21treat.Genes.DESeq2.txt')
# Add "symbol" column
de.genes <- de.genes %>% mutate(symbol = rownames(de.genes))
# Remove genes without differential expression statistics

de.genes <- de.genes[ !is.na(de.genes[, "padj"]), ]
# Select the most significant differentially expressed genes
de.genes <- de.genes[de.genes[, "padj"] < 0.05, ] # FDR cutoff 0.05
# Merge differentially expressed genes with genomic coordinates
de.genes <- left_join(de.genes, hg19_symbols, by = c('symbol' = 'symbol'))
# Remove rows with NAs
de.genes <- de.genes[complete.cases(de.genes),]
# Make GRanges object for DE genes
de.genes.gr <- GRanges(de.genes$chr, IRanges(start = de.genes$start,
                                             end = de.genes$end))

# Find overlaps
olaps <- findOverlaps(sig.regions.gr, de.genes.gr)
olaps

Hits object with 977 hits and 0 metadata columns:
      queryHits subjectHits
      <integer>  <integer>
[1]           5         419
[2]           5         597
[3]          12        5685
[4]          12        5686
[5]          13        1043
...           ...         ...
[973]        2015        4823
[974]        2017        5525
[975]        2018           2
[976]        2018        5108
[977]        2022         127
-----
queryLength: 2027 / subjectLength: 6127
```

We can see that there are 977 overlaps between the DE genes and our significant regions. To test if this amount of overlap is significantly different from what can be expected by chance, we can perform a permutation test by testing the overlap of the DE genes with randomly selected regions from the genome. We can use multiHiCcompare's built in permutation test for checking the enrichment of genomic features:

```
# use multiHiCcompare's permutation test function
p.value <- perm_test(rao2017, de.genes.gr, p.adj_cutoff = 10^-15,
                    logfc_cutoff = 1, num.perm = 1000)
p.value
[1] 0.000999001
```

We can see that the DE genes are significantly enriched in our DIRs that were detected by multiHiCcompare (p-value = 9.990E-04).

Functional enrichment of genes overlapping DIRs—Given the significant overlap between DE genes and DIRs, it may be of interest to test whether all genes overlapping DIRs are enriched in any canonical pathway or gene ontology annotation category. This can be done using the ROntoTools R package (Voichita and Draghici., 2012).

First, we need to get the genomic locations of the genes including strand information:

```
library(clusterProfiler) # BiocManager::install("clusterProfiler")
library(DOSE) # BiocManager::install("DOSE")
library(ROntoTools) # BiocManager::install("ROntoTools")
library(graph) # BiocManager::install("graph")

# Make GRanges out of all genes
hg19_symbols.gr <- makeGRangesFromDataFrame(hg19_symbols,
                                             seqnames.field = 'chr',
                                             start.field = 'start',
                                             end.field = 'end',
                                             strand.field = 'strand',
                                             keep.extra.columns = TRUE)

# Overlap genes with DIRs defined previously
olap <- findOverlaps(sig.regions.gr, hg19_symbols.gr)
```

Next, we need to create a named vector for the number of times each region was detected as significantly interacting. The names for this vector will be the Entrez gene IDs. This vector will be used for the pathway enrichment to walk down the list of genes overlapping most-to-least frequently detected DIRs:

```
# Create "gene_counts" data.frame with the column for count and gene symbol
genes_olap <- olap %>% as.data.frame %>% group_by(queryHits) %>%
  mutate(genes = hg19_symbols.gr[elementMetadata$subjectHits]) %>%
  dplyr::select(queryHits, genes) %>% distinct()

tmp <- sig.regions %>% dplyr::select(count, avgLogFC, avgP.adj) %>% mutate(id =
1:nrow(sig.regions))
gene_counts <- left_join(genes_olap, tmp, by = c('queryHits' = 'id'))

# Convert gene symbols into entrez ID
entrez <- bitr(gene_counts$genes, fromType = 'SYMBOL', toType = 'ENTREZID', OrgDb =
'org.Hs.eg.db')

# Join the Entrez ID to the counts
gene_counts <- left_join(gene_counts, entrez, by = c('genes' = 'SYMBOL'))
# Remove unmapped entries
gene_counts <- gene_counts[complete.cases(gene_counts), ]

# Make the named vector of fold changes and pvalues for genes
fc <- gene_counts$avgLogFC
names(fc) <- paste0('hsa:', gene_counts$ENTREZID)
pv <- as.numeric(gene_counts$avgP.adj)
names(pv) <- paste0('hsa:', gene_counts$ENTREZID)

# Load KEGG pathways
kpg <- keggPathwayGraphs("hsa", updateCache = TRUE, verbose = FALSE)
# set edge weights
kpg <- setEdgeWeights(kpg, edgeTypeAttr = "subtype",
                      edgeWeightByType = list(activation = 1, inhibition = -1,
expression = 1, repression = -1),
                      defaultWeight = 0)
```

Now we can plug the fc (fold-change) and pv (p-value) vectors into ROntoTools pathway analysis:


```
# Set node weights
kpg <- setNodeWeights(kpg, weights = alpha1MR(pv), defaultWeight = 1)
# Perform pathway analysis
peRes <- pe(x = fc, graphs = kpg, ref = paste0('hsa:',
as.character(hg19_symbols$entrez)), nboot = 200, verbose = FALSE)
# Prepare results table
kpn <- keggPathwayNames("hsa")
table1 <- head(Summary(peRes, pathNames = kpn, totalAcc = FALSE, totalPert = FALSE,

pAcc = FALSE, pORA = FALSE, comb.pv = NULL, order.by = "pPert"), n = 15)
table1$pPert <- round(table1$pPert, digits = 3)
table1$pPert.fdr <- round(table1$pPert.fdr, digits = 3)
```

Here we can see the results of the Pathway analysis (Table 1). These pathways do not make much sense in the context of the auxin vs. normal experiment. This could be because auxin degrades looping throughout the genome and does not target any specific function of the cell. This would explain the unrelated pathways found to be enriched.

We can also plot the pathways using ROntoTools to visualize the propagation across a specific pathway. Select a pathway name of interest, for example, “Cytokine-cytokine receptor interaction” pathway (path:hsa04060). We then create the plot as follows (figure omitted due to large size).

```
# Select pathway
p <- peRes@pathways[["path:hsa04972"]]
# Create graph
g <- layoutGraph(p@map, layoutType = "dot")
graphRenderInfo(g) <- list(fixedsize = FALSE)
edgeRenderInfo(g) <- peEdgeRenderInfo(p)
nodeRenderInfo(g) <- peNodeRenderInfo(p)
# Plot the graph
renderGraph(g)
```

Overlap enrichment between TAD boundaries and DIRs—Another plausible hypothesis to test is the overlap between DIRs and boundaries of topologically associated domains (TADs).

We will first need to identify the TADs for the datasets being used. We can use the TopDom R script for TAD identification. TopDom can be downloaded from here: <http://zhoulab.usc.edu/TopDom/>. To use TopDom, the user will need to download the R script to the working directory and source it in the R session. Note: TADs are typically called using Hi-C data at resolutions of 50KB or higher, however for simplicity here we continue to use the 100KB data. If the user plans to perform an analysis using TADs, he should call them at 50KB resolution or higher.


```
# Download TopDom
wget http://zhoulab.usc.edu/TopDom/code/TopDom_v0.0.2.zip
unzip TopDom_v0.0.2.zip

# Source TopDom script
source('TopDom_v0.0.2.R')
```

Next, we will create the matrix file necessary for TopDom. TopDom requires an $N \times (N + 3)$ matrix in a text file. We can create this file for chromosome 1 as follows:

```
# Convert sparse matrix read in at beginning of tutorial to a full matrix
mat <- sparse2full(sample_list[[1]][chr == 1, c('region1', 'region2', 'IF')])
# Create 3 extra columns necessary for TopDom
bed <- data.frame(chr = 'chr1', start = colnames(mat),
                  end = as.numeric(colnames(mat)) + resolution(rao2017))
# Merge 3 columns with full matrix
mat <- cbind(bed, mat)
# Write as a text file for input into TopDom
write_tsv(mat, path = 'chr1.matrix', col_names = FALSE)
```

The user should now have a text file containing the $N \times (N + 3)$ contact matrix for chromosome 1 in the file chr1.matrix. Now we can input the matrix into TopDom and get the TAD boundaries:

```
TADs <- TopDom(matrix.file="chr1.matrix", window.size=5)
```

The results contain a BED file indicating the positions of the gaps, domains, and boundaries. We will pull out the locations of the boundaries to check if the DIRs are enriched within them:

```
# Pull out the bed file from the TopDom results with boundary locations
boundaries <- TADs$bed
# Subset to only boundaries
boundaries <- boundaries[boundaries$name == "boundary",]

# Convert to GRanges
boundaries <- makeGRangesFromDataFrame(boundaries,
                                       seqnames.field = 'chrom',
                                       start.field = 'chromStart',
                                       end.field = 'chromEnd',
                                       keep.extra.columns = TRUE)
```

Similarly, we prepare a list of DIRs on chromosome 1:

```
# Make GRanges object for DIRs from `counts` object created with the topDIRs function
chr1.dir <- counts[counts$chr == 'chr1', ]
chr1.dir <- makeGRangesFromDataFrame(chr1.dir,
                                     seqnames.field = 'chr',
                                     start.field = 'start',
                                     end.field = 'end',
                                     keep.extra.columns = TRUE)

# Find overlaps between boundaries and DIRs
olaps <- findOverlaps(chr1.dir, boundaries)
olaps

Hits object with 7 hits and 0 metadata columns:
      queryHits subjectHits
      <integer>  <integer>
[1]          9          11
[2]         58          10
[3]         82          27
[4]        113          15
[5]        119          27
[6]        120          27
[7]        137           9
-----
queryLength: 168 / subjectLength: 28
```

Next, we perform a permutation test similar to the one performed in the previous section. This will test for enrichment of DIRs within the TAD boundaries:

```
# subset rao2017 Hicexp object to only chr1
chr1.rao2017 <- rao2017
slot(rao2017, "comparison") <- results(rao2017)[chr == 1, ]

# perform permutation test
p.value <- perm_test(rao2017, boundaries, p.adj_cutoff = 0.01,
                    logfc_cutoff = 1, num.perm = 1000)
p.value

[1] 0.6183816
```

The DIRs do not seem to be enriched within TAD boundaries. This could be due to the simplifications we took for this tutorial. TADs should be called at resolutions of 50KB or higher so it is possible our TAD boundaries are not as accurate as they should be. Additionally, it is possible that the changes induced by auxin do not target TAD boundaries but instead target smaller loop boundaries within the TADs.

Overlap between DIRs and binding sites—The auxin treatment used in (Rao et al., 2017) was noted to destroy the RAD21 complex. Thus, our DIRs may correspond to changes at RAD21 binding sites.

We will need to download the location information for RAD21 binding sites for HCT-116 cells using the following link: http://dc2.cistrome.org/api/downloads/eyJpZCI6IjQ2MjA3In0:1g7Oxj:qCQcO9uSS36i7LdGEQGz7KAeZ_gb

Save the 46207_peaks.bed file into the working directory. Then we can read the file into R:

```
rad21 <- read.table('46207_peaks.bed')

head(rad21)

      V1      V2      V3      V4      V5
1 chr1 10149 10295 peak1 6.70350
2 chr1 16184 16359 peak2 13.33365
3 chr1 91401 91609 peak3 42.12526
4 chr1 104859 105106 peak4 57.36115
5 chr1 181917 182071 peak5 11.32777
6 chr1 186798 187100 peak6 10.35612
```

This is a standard BED file. We will need to convert it into a GRanges object so that we can input these locations into the permutation test function:

```
# convert X and Y chr names into 23 and 24 to correspond with multiHiCcompare results
rad21$V1 <- sub("X", "23", rad21$V1)
rad21$V1 <- sub("Y", "24", rad21$V1)

# convert to GRanges
rad21 <- GRanges(rad21$V1, IRanges(start = rad21$V2, end = rad21$V3))

# input into permutation test
perm_test(rao2017, rad21, p.adj_cutoff = 10^-15, logfc_cutoff = 1, num.perm = 1000)

[1] 0.01898102
```

RAD21 sites are significantly enriched in the DIRs, confirming the published observations (Rao et al., 2017). This indicates that auxin destroying RAD21 does lead to a change in chromatin interactions. Depending on the experimental conditions for the data being analyzed the user may want to try a similar test on other binding sites. Additionally, CTCF sites are typically correlated with TAD boundaries so they may be a genomic feature which the user would want to check for enrichment in DIRs.

Summary

This protocol describes the R-based workflow for the high-level analysis and interpretation of a comparative analysis of multiple Hi-C datasets. The main functionality is provided by the multiHiCcompare R package. The workflow describes the joint normalization of multiple Hi-C datasets, the detection of differentially interacting regions, and the downstream interpretation of the results. The steps mainly involve the use of the command line and Bioconductor R packages and should be generalizable to any operating system capable of running R.

Basic Protocol 2: Comparative analysis of Hi-C data using diffHic

Introduction

In this protocol, we provide a brief tutorial of diffHic, which is an alternative software for identifying differentially interacting chromatin regions (DIRs) (Lun and Smyth, 2015). Here we use the same example data that were used in Protocol 1. The diffHic package offers

several advantages. The tool is able to recognize patterns of restriction enzyme cutting for a more efficient division of the genome. It also provides functions to reduce artifacts and trend biases in the data, as well as offers a variety of statistical methods for DIR identification. The diffHic R package provides a complete pipeline from raw sequencing data processing to alignment to DIRs identification. However, due to its incompatibility with other popular Hi-C raw data processing packages, users are required to process the raw data and align the reads themselves instead of utilizing the pre-processed data deposited in public repositories. More details on the statistical methods and advanced analyses can be found in the edgeR (Robinson and Smyth, 2008) and diffHic (Lun and Smyth, 2015) manuals on Bioconductor.

Necessary Resources

Hardware—A computer with internet access, at least 2TB of free hard drive space (if the user wishes to run the full pipeline starting with raw data). A computing cluster is highly recommended for performing the alignment process.

Software—The R (version 3.5.0) programming environment, a Unix based command-line interface (e.g., bash on Linux), bowtie2 (version 2.3), cutadapt (version 1.18), Biopython (version 1.72), Pysam (version 0.15), and a web browser.

Files—The fastq files for the data to be used. For our example, we will cover downloading the necessary files in the following section. Or if the user wishes to skip the alignment steps, only the .h5 files provided here are required: (URL to be provided after submission)

Downloading and processing data

Here we use the same datasets that were used in Protocol 1: two normal HCT-116 cells and two HCT-116 cells treated with auxin for 6 hours. The fastq files can be downloaded using the SRADB package. Note that downloading the raw Hi-C data will require a large amount of storage (~2TB) and alignment will take a significant amount of time. To skip the raw data processing steps proceed to the Starting with .h5 files section.

```
# install.packages("BiocManager")
library(SRADb) # BiocManager::install('SRADB')
library(diffHic) # BiocManager::install('diffHic')
library(BSgenome.Hsapiens.UCSC.hg19) #
BiocManager::install('BSgenome.Hsapiens.UCSC.hg19')

# Get required file for SRADB for the first time.
# After that, we need to set the path to the downloaded sqlfile if we want to use SRADB
to download data again.
sqlfile <- getSRADBFile()
sqlfile <- file.path("SRAMetadb.sqlite")
sra_con <- dbConnect(SQLite(),sqlfile)
```

Now we download the fastq files from the short read archive.

```
# Get data, input could be the whole experiment or a specific run.
# Save results to individual folders (make folder if it does not exist).
getSRAfile( c("SRX3222724"), sra_con, fileType = 'fastq',
            makeDirectory = TRUE, destDir = 'HIC001')

getSRAfile( c("SRX3222725"), sra_con, fileType = 'fastq',
            makeDirectory = TRUE, destDir = 'HIC002')

getSRAfile( c("SRX3276107"), sra_con, fileType = 'fastq',
            makeDirectory = TRUE, destDir = 'HIC008')

getSRAfile( c("SRX3276108"), sra_con, fileType = 'fastq',
            makeDirectory = TRUE, destDir = 'HIC009')
```

Next, we can get the SRA information including the names of the data files.

```
# Get the names of runs and replicates for later data processing
hc1 <- getSRAinfo("SRX3222724", sra_con)$run
hc2 <- getSRAinfo("SRX3222725", sra_con)$run
hc3 <- getSRAinfo("SRX3276107", sra_con)$run
hc4 <- getSRAinfo("SRX3276108", sra_con)$run
exp <- paste0("HIC00",c(1,2,8,9))
exp.run <- list(hc1,hc2,hc3,hc4)
```

After downloading the fastq files, we can align the reads to the reference genome. Bowtie2, cutadapt, and python with the Biopython and pysam packages installed are required to run this step. Additionally, the presplit_map.py script should be copied to the working directory. The presplit_map.py script can be found by running the following command in R.

```
system.file ("python", "presplit_map.py", package="diffHic", mustWork=TRUE)
```

We will need to extract any gzipped files and download the chromosome 1 bowtie index for hg19.

```
# Extract all the gzip files
gunzip */*.gz
# Bowtie index for chromosome 1 of human genome
wget ftp://hgdownload.cse.ucsc.edu/goldenPath/hg19/chromosomes/chr1.fa.gz
# Make "hg19" folder to store genome index
mkdir hg19
# Build index
gunzip chr1.fa.gz -d hg19
cd hg19
bowtie-build --threads 4 chr1.fa hg19chr1 # 4 is the number of processors
cd ..
```

We are now ready to start the alignment process. Note this should be performed on a computing cluster if one is available as it is very time-consuming.

```

# Fill in the path to the hg19 index files at working_dir/hg19
export BOWTIE2_INDEXES=/path/to/my/bowtie2/databases/hg19

# Use diffHic presplit_map.py to generate bam files;
# cores = is the maximum number of python scripts run at the same time.
# Note that users can use the -p option for bowtie and -j option for cutadapt
# to increase the number of parallel processes in the following code.
# The python script multiplies the number of threads assigned to bowtie and cutadapt
# and should not exceed the number of cores.

cores=12
for filename in HIC001/*_1.fastq; do
  python presplit_map.py -G hg19chr1 -1 ${filename%_1.fastq}_1.fastq -2
  ${filename%_1.fastq}_2.fastq --cmd "bowtie2 -p 4" --cut "cutadapt -j 2" --sig GATC -o
  ${filename%_1.fastq}.bam &
  background=( $(jobs -p) )
  if (( ${#background[@]} == cores )); then
    wait -n
  fi
done

for filename in HIC002/*_1.fastq; do
  python presplit_map.py -G hg19chr1 -1 ${filename%_1.fastq}_1.fastq -2
  ${filename%_1.fastq}_2.fastq --cmd "bowtie2 -p 4" --cut "cutadapt -j 2" --sig GATC -o
  ${filename%_1.fastq}.bam &
  background=( $(jobs -p) )
  if (( ${#background[@]} == cores )); then
    wait -n
  fi
done

for filename in HIC008/*_1.fastq; do
  python presplit_map.py -G hg19chr1 -1 ${filename%_1.fastq}_1.fastq -2
  ${filename%_1.fastq}_2.fastq --cmd "bowtie2 -p 4" --cut "cutadapt -j 2" --sig GATC -o
  ${filename%_1.fastq}.bam &
  background=( $(jobs -p) )
  if (( ${#background[@]} == cores )); then
    wait -n
  fi
done

for filename in HIC009/*_1.fastq; do
  python presplit_map.py -G hg19chr1 -1 ${filename%_1.fastq}_1.fastq -2
  ${filename%_1.fastq}_2.fastq --cmd "bowtie2 -p 4" --cut "cutadapt -j 2" --sig GATC -o
  ${filename%_1.fastq}.bam &
  background=( $(jobs -p) )
  if (( ${#background[@]} == cores )); then
    wait -n
  fi
done

```

After executing the above script, the user should have four folders containing the bam input files for diffHic.

Building the interaction matrix

diffHic was designed to use the recognition pattern for the restriction enzyme used in the Hi-C data generation to effectively divide the genome into fragments. The MboI restriction enzyme which cuts at the GATC pattern was used to generate the data for this example. To use diffHic on an alternate data source, the user will need to determine the restriction enzyme used and its cut site. The DNA fragments are obtained using cutGenome() on the human reference genome.


```
# Digest genome using MboI resction enzyme
hs.frag <- cutGenome(BSgenome.Hsapiens.UCSC.hg19, "GATC", 4)
hs.frag

GRanges object with 7227606 ranges and 0 metadata columns:
      seqnames      ranges strand
      <Rle>      <IRanges> <Rle>
[1]      chr1      1-11163      *
[2]      chr1    11160-12414      *
[3]      chr1    12411-12464      *
[4]      chr1    12461-12689      *
[5]      chr1    12686-12832      *
...      ...      ...      ...
[7227602] chrUn_gl000249 38117-38206      *
[7227603] chrUn_gl000249 38203-38288      *

[7227604] chrUn_gl000249 38285-38436      *
[7227605] chrUn_gl000249 38433-38502      *
[7227606] chrUn_gl000249 38499-38502      *
-----
seqinfo: 93 sequences from hg19 genome
```

Next, we can use the function `pairParam()` to generate the reference for `diffHic`. In this tutorial, we will only investigate the DIRs within chromosome 1 by setting the `restrict` parameter of `pairParam()`. Users can easily expand their analysis to other chromosomes by modifying our example code.

```
hs.param <- pairParam(hs.frag, restrict = 'chr1')
hs.param

Genome contains 7227606 restriction fragments across 93 chromosomes
No discard regions are specified
Read extraction is limited to 1 chromosome
No cap on the read pairs per pair of restriction fragments
```

To proceed, we need to create `.h5` files that serve as input for `diffHic`. We can execute the command `preparePairs()` on the bam files to create the corresponding `.h5` files. Typically, we can execute the command `preparePairs` to create an `.h5` file for each bam file (of a run) and then use the command `prunePairs` to remove artifacts that occurred in the experiments. If a sample has multiple runs (multiple fastq and corresponding `.bam` files), we will get multiple `.h5` files one for each run. These `.h5` files can be combined using `mergePairs()` to create a single interaction matrix for a sample.

```

# Data processing for each dataset
diagnostics <- list()
counted <- list()
for (i in 1:4) {
  cur.exp <- exp[i]
  for (run in exp.run[[i]]) {
    run.name <- paste0(cur.exp, "/", run)
    diagnostics[[run.name]] <- preparePairs(paste0(run.name, ".bam"),
                                           hs.param, file = paste0(run.name, ".h5"),
                                           dedup = TRUE, minq = 10)

    counted[[run.name]] <- prunePairs(paste0(run.name, ".h5"),
                                     hs.param, file.out =
paste0(run.name, "_trimmed.h5"),
                                     max.frag = 600, min.inward = 1000,
                                     min.outward = 25000)
  }
  mergePairs(files = paste0(cur.exp, "/",
                             paste0(exp.run[[i]], "_trimmed.h5")),
             paste0(cur.exp, ".h5"))
}

```

The interaction files (one interaction file per sample) can be combined into a single object using `squareCounts()`. The boundaries of each bin are rounded to the nearest restriction fragment size. The bin size of 1Mb is often used to get a reasonable number of interactions between bin-pairs.

```

# Load the .h5 files
input <- c('HIC001.h5', 'HIC002.h5', 'HIC008.h5', 'HIC009.h5')
bin.size <- 1e6 # set the bin size
data <- squareCounts(input, hs.param, width=bin.size, filter=1)
data

class: InteractionSet
dim: 26796 4
metadata(2): param width
assays(1): counts
rownames: NULL

```

```

rowData names(0):
colnames: NULL
colData names(1): totals
type: ReverseStrictGInteractions
regions: 3041

```

Starting with .h5 files

If the user wishes to skip the alignment steps, he may start here with the .h5 files. Download the .h5 files to the working directory (URL to be provided after submission). Then load the data as follows. If the user has already aligned and processed the data from fastq files then please skip this section.

```
# Load necessary packages if not done so already
library(diffHic)
library(BSgenome.Hsapiens.UCSC.hg19)

# Digest genome using MboI resction enzyme
hs.frag <- cutGenome(BSgenome.Hsapiens.UCSC.hg19, "GATC", 4)
# Restrict to chr1
hs.param <- pairParam(hs.frag, restrict = 'chr1')

# Load the .h5 files
input <- c('HIC001.h5', 'HIC002.h5', 'HIC008.h5', 'HIC009.h5')
# Set the bin size
bin.size <- 1e6
data <- squareCounts(input, hs.param, width=bin.size, filter=1)
```

Data filtering and normalization

The user can filter out bin-pairs with low counts using the average log count per million (logCPM). Here we set the threshold to the average of a theoretical bin-pair with counts of 5 in all datasets. The bin-pairs with an average lower than the threshold are considered uninteresting and are thus removed.

```
library(edgeR) # BiocManager::install("edgeR")
# Get the average logCPM
ave.ab <- aveLogCPM(asDGEList(data))

# Plot histogram of avg logCPM
hist(ave.ab, xlab="Average abundance", col="grey80", main="")

# Set which entries to keep
keep <- ave.ab >= aveLogCPM(5, lib.size=mean(data$totals))

# Backup original data
original.data <- data
# Remove filtered entries
data <- data[keep,]
```

After filtering out bin-pairs with low counts, we should normalize the data from different datasets to avoid trend biases. We will demonstrate trend biases by using MA plots of the data before and after normalization (Fig. 6).

```
library(csaw) # BiocManager::install("csaw")

# Calculate A
ab <- aveLogCPM(asDGEList(data))
# Order avg logCPM
```

```

o <- order(ab)
# Calculate counts per million
adj.counts <- cpm(asDGEList(data), log=TRUE)
# Calculate M
mval <- adj.counts[,3]-adj.counts[,2]

# Plot MA plot
smoothScatter(ab, mval, xlab="A", ylab="M", main="Treated (1) vs. Normal (2)")

# Fit Loess curve to MA plot
fit <- loessFit(x=ab, y=mval)

# Add Loess fit to MA plot
lines(ab[o], fit$fitted[o], col="red")

```

normOffsets() can be used on the data to calculate the offsets for our datasets. After applying the normOffsets() on the data, the trend biases disappear in the new MA plot (Fig. 6B).

```

# Calculate offsets

data <- normOffsets (data, type="loess", se.out=TRUE)

```

However, bin-pairs near the diagonal of the interaction matrix (short range interactions) usually have much larger counts compared to the long-range interactions. Therefore, for more accurate normalization, offsets for near diagonal bin-pairs should be calculated separately to avoid a loss of information for the other bin-pairs.

```

# Filter bins near the diagonal
neardiag <- filterDiag(data, by.dist=1.5e6)
# Create offsets matrix with the same dimension as data
nb.off <- matrix(0, nrow=nrow(data), ncol=ncol(data))
# Calculate offsets
nb.off[neardiag] <- normOffsets(data[neardiag,], type="loess", se.out=FALSE)
nb.off[!neardiag] <- normOffsets(data[!neardiag,], type="loess", se.out=FALSE)
# Update the offset matrix
assay(data, "offset") <- nb.off

# Offsets are applied to log2 of count data.
# 0.5 is added to the counts to prevent an error if count = 0
# Offsets are calculated using log10 so they are divided by log(2) to convert to base 2.
adj.counts <- log2(assay(data) + 0.5) - assay(data, "offset")/log(2)
# Calculate M values
mval <- adj.counts[,3]-adj.counts[,2]

# Plot the MA plot
smoothScatter(ab, mval, xlab="A", ylab="M", main="Treated (1) vs. Normal (2)")

# Fit the Loess curve
fit <- loessFit(x=ab, y=mval)

# Plot the Loess fit
lines(ab[o], fit$fitted[o], col="red")

```

Detecting differential interactions and visualization

The differential analysis functions of diffHiC are built on edgeRs statistical framework. In diffHiC, variability is modeled by estimating the dispersion parameters of the negative-binomial (NB) distribution and quasi-likelihood (QL) dispersion. This is used for hypothesis testing to detect DIRs.

First, we need to specify the design matrix that describes the experimental setup. In the code below, we first specify two groups (normal versus treated) and then convert the data into a DGEList object for analysis with edgeR. The NB dispersion can be estimated using the command estimateDisp(). The plot of the biological coefficient of variation is displayed in Fig. 7A.

```
# Set up design matrix
design <- model.matrix(~factor(c("Normal", "Normal", "Treated", "Treated")))
colnames(design) <- c("Intercept", "Treated")

# Create DGEList
y <- asDGEList(data)

# Estimate the dispersion
y <- estimateDisp(y, design)

# Plot the biological coefficient of variation
plotBCV(y)
```

We can now fit a general linear model (GLM) to the data and plot the QL dispersion (Fig. 7B) for our data using the following code.

```
# Fit GLM
fit <- glmQLFit(y, design, robust=TRUE)

# Plot QL dispersion
plotQLDisp(fit)
```

After dispersion estimation, `glmQLFTest()` can be used to perform a quasi-likelihood F-test to identify bin-pairs with significant differences. The output of `glmQLFTest()` includes logFC, p-values and FDR-corrected p-values.

```
# Perform F test
result <- glmQLFTest(fit, coef=2)
# Display results
topTags(result)
```

	Coefficient: Treated					
	logFC	logCPM	F	PValue	FDR	
1709	-1.312196	7.916977	500.6551	3.082966e-16	8.261115e-12	
8122	-1.158629	7.873687	393.7927	3.519769e-15	4.127811e-11	
526	-1.184617	8.465543	381.1573	4.889552e-15	4.127811e-11	
7996	-1.201872	7.416843	372.4990	6.161832e-15	4.127811e-11	
6212	1.277441	5.893027	328.6729	2.161274e-14	9.269023e-11	
1251	1.875915	3.648410	327.1167	2.266207e-14	9.269023e-11	
2014	-1.021491	8.143626	324.9544	2.421375e-14	9.269023e-11	
4849	-1.026293	8.177141	311.2873	3.716517e-14	1.184421e-10	
134	-1.185305	6.945935	309.1678	3.978128e-14	1.184421e-10	
6213	1.150091	6.390913	292.1984	6.970974e-14	1.867942e-10	

The interaction matrices of the differentially interacting regions can be plotted using the `plotPlaid()` function, which requires boundaries from processed data and the raw data from the .h5 files as input (Fig. 8).


```
# Get order of p-values
o.r <- order(result$table$PValue)
# Pick difference to plot
chosen <- o.r[1]
# Get genomic region for plotting
chosen.a1 <- anchors(data[chosen], type="first")
chosen.a2 <- anchors(data[chosen], type="second")

expanded1 <- resize(chosen.a1, fix="center", width=bin.size*5)
expanded2 <- resize(chosen.a2, fix="center", width=bin.size*5)
```

The color of each pixel in this plot is correlated to the count. To prevent large counts from dominating the plot, all counts bigger than the cap set below are set to this maximum value. The cap for the treated sample is calculated from the normal cap by multiplying it with the ratio of total counts from the datasets.

```
cap.wt <- 200 # Set cap for normal
cap.t <- cap.wt*data$totals[3]/data$totals[1] # Set cap for treated

# Set up side by side plot of matrix
par(mfrow=c(1,2))
# Plot matrix for normal samples
plotPlaid(input[1], first=expanded1, second=expanded2, max.count=cap.wt,
width=5e4, param=hs.param, main="Normal")
rect(start(chosen.a1), start(chosen.a2), end(chosen.a1), end(chosen.a2))

# Plot matrix for treated samples
plotPlaid(input[3], first=expanded1, second=expanded2, max.count=cap.t,
width=5e4, param=hs.param, main="Treated")
rect(start(chosen.a1), start(chosen.a2), end(chosen.a1), end(chosen.a2))
```

Basic Protocol 3: Comparative analysis of Hi-C data using FIND

Introduction

diffERential chromatin INteractions Detection using a spatial Poisson process (FIND) is another R package for comparing Hi-C data (Djekidel et al., 2018). FIND was developed with the analysis of high-resolution Hi-C data in mind. It uses a spatial Poisson process which considers local spatial dependencies between interacting regions of the chromatin. FIND was designed to detect differential chromatin interactions that are significantly different in their interaction frequency. In this protocol, we will perform an example analysis using FIND on the Rao 2017 data (Rao et al., 2017) that were used in Protocol 1.

Necessary Resources

Hardware—A computer with internet access, at least 35GB of free hard drive space, and 8GB of RAM.

Software—The R (version 3.5.0) programming environment, the FIND R package (Version 0.99), a Unix based command-line interface, and a web browser.

Files—The .hic files used in Basic Protocol 1 will be used again in this protocol.

Installing FIND

FIND's development page is on bitbucket here: <https://bitbucket.org/nadhir/find>. We can download the source R package from the downloads section of the page:

```
wget https://bitbucket.org/nadhir/find/downloads/FIND_0.99.tar.gz
```

We now need to install FIND in R. First make sure all dependencies are installed and then install FIND from the source package:

```
# Install dependencies
install.packages(c("Rcpp", "RcppEigen", "Matrix", "bigmemory", "data.table",
                  "doParallel", "quantreg", "png", "dplyr"))
BiocManager::install(c("HiTC", "zlibbioc"))

# Install FIND from source
install.packages("FIND_0.99.tar.gz", repos = NULL, type="source")

library(FIND)
```

Extracting the data

Now we will need to obtain the data we will use for this example. Assuming the user has already downloaded the .hic files used in protocol 1 we will just need to extract the matrices at 5KB resolution. We will focus this analysis on only chromosome 18 for demonstration purposes. We will also extract the Knight-Ruiz normalized matrices from the files, as recommended by the FIND paper (Djekidel et al., 2018):

```
./straw KR GSM2795535_Rao-2017-HIC001_30.hic 18 18 BP 5000 >
HIC001/HIC001.KR.chr18.5000.txt
./straw KR GSM2795536_Rao-2017-HIC002_30.hic 18 18 BP 5000 >
HIC002/HIC002.KR.chr18.5000.txt
./straw KR GSM2809539_Rao-2017-HIC008_30.hic 18 18 BP 5000 >
HIC008/HIC008.KR.chr18.5000.txt
./straw KR GSM2809540_Rao-2017-HIC009_30.hic 18 18 BP 5000 >
HIC009/HIC009.KR.chr18.5000.txt
```

We can now read the data into R.

```
library(readr) # BiocManager::install("readr")

hic001 <- read_tsv("HIC001/HIC001.KR.chr18.5000.txt", col_names = FALSE)
hic002 <- read_tsv("HIC002/HIC002.KR.chr18.5000.txt", col_names = FALSE)
hic008 <- read_tsv("HIC008/HIC008.KR.chr18.5000.txt", col_names = FALSE)
hic009 <- read_tsv("HIC009/HIC009.KR.chr18.5000.txt", col_names = FALSE)
```

Use FIND to compare datasets

FIND operates on dgCMMatrix objects so we will need to convert our sparse matrices into this format.

```

library(Matrix) # BiocManager::install("Matrix")
library(mvtnorm) # BiocManager::install("mvtnorm")
library(rasterVis) # BiocManager::install("rasterVis")
library(gridExtra) # BiocManager::install("gridExtra")
library(HiTC) # BiocManager::install("HiTC")
library(edgeR) # BiocManager::install("edgeR")
library(ggsci) # BiocManager::install("ggsci")
library(HiCcompare) # BiocManager::install("HiCcompare")

# Convert sparse matrices to full
hic001 <- sparse2full(hic001)
hic002 <- sparse2full(hic002)
hic008 <- sparse2full(hic008)
hic009 <- sparse2full(hic009)

# Convert to dgCMatix format
hic001 <- as(hic001, "dgCMatix")
hic002 <- as(hic002, "dgCMatix")
hic008 <- as(hic008, "dgCMatix")

hic009 <- as(hic009, "dgCMatix")

# Make a list of the matrices for the two groups
control <- list(hic001, hic002)
auxin <- list(hic008, hic009)

```

We are now ready to enter the data into FIND. However, due to the long running time even on the short chromosome 18 (>500 hours), the output is not provided in this tutorial.

```

DCis <- getDCIs_fromMat(control, auxin, windowSize = 3,
                        alpha = 0.7, method = "hardCutoff",
                        qvalue = 1e-06, isrOP_qval = FALSE)

```

Guidelines for Understanding Results

Basic Protocol 1

This protocol was designed in a way so that even a novice user can work through the entire Hi-C experiment analysis process. The protocol focuses on using the multiHiCcompare R package. The downstream analysis section focuses on ways to test the differentially interacting regions for the enrichment of genomic features and visualization of the results. The plotting functions provided by multiHiCcompare can be used to assess the results of the procedure. If the MD plots do not show the data centered and symmetric around 0, there may have been issues in the normalization steps, likely due to the sparsity of the data. Additionally, the distribution of differential regions on the Manhattan plots might indicate faulty results. For example, DIRs clustering near centromeres or telomeres might be due to artifacts resulting from difficulties sequencing these regions of the genome. Depending on the specifics of the comparison being performed the user should expect to see enrichment of

related genomic features in the differentially interacting regions or enrichment of genes and pathways thought to play a role in differentiating the experimental groups.

Basic Protocol 2

The goal of this protocol is to guide the user through the process of aligning raw Hi-C data into count matrices and detecting differentially interacting regions using diffHiC. The final result from diffHiC is a list of DIRs. These DIRs can be visualized using diffHiC's functions or be converted for use in multiHiCcompare's visualization functions. The list of DIRs from diffHiC should closely resemble the results of multiHiCcompare and thus many of the downstream analyses shown in protocol 1 can also be applied to the results of diffHiC. The package also provides additional options for checking the quality of the data, removing artifacts, and correcting for biases.

Basic Protocol 3

The results of FIND are similar to those of multiHiCcompare. FIND will return a list of regions which were detected as differentially interacting. These regions can then be used in downstream analyses similar to what was shown in Basic Protocol 1. FIND was designed to be used on very high-resolution Hi-C data. Since FIND makes use of the spatial dependence between nearby regions it may not be applicable to low resolution datasets. Another consideration for working with FIND is its long run times. Parallel processing should be used if possible. If more immediate results are needed, or lower resolution data is being analyzed, it is recommended to follow protocol 1 or protocol 2 instead.

Commentary

Background Information

Hi-C techniques evolved out of the original chromatin conformation capture (3C) methods (Dekker et al., 2002). 3C methods were limited to only capturing the 3D structure of a small subset of the genome at one time. These methods were developed further into 4C, 5C and, finally, Hi-C, which allows for an all vs. all capture of the chromatin interactions across the entire genome (Lieberman-Aiden et al., 2009). Briefly, genomic DNA is cross-linked, stabilizing spatially adjacent chromatin regions. The ends of the combined chromatin fragments are cut with an enzyme, joined, cross-links are removed, and the joined chromatin fragments are sequenced. These joined chromatin fragments represent genomic regions interacting in close proximity to each other. Mapping them to the reference genome identifies genomic coordinates of interacting chromatin regions. The number of mapped fragments connecting a pair of regions corresponds to the strength of interactions between them.

Initial studies focused on analyzing individual Hi-C datasets. Consequently, methods for normalizing (removing biases in) individual matrices were developed. They can be broadly divided into two general approaches: explicit and implicit bias correction methods. The explicit bias models consider factors such as mappability, GC content, and fragment length (Yaffe and Tanay, 2011). The implicit approaches, also known as matrix balancing, iterative correction algorithms, are based on the assumption of "equal visibility." The "equal

visibility” approach assumes that since we are interrogating the entire interaction space in an unbiased manner, each fragment/bin should be observed approximately the same number of times in the experiment (interpreted as the sum of the genome-wide row/column in the interaction matrix). Some of the well-known adaptations of matrix balancing algorithms include Knight-Ruiz (KR) normalization (Knight and Ruiz, 2012) and Iterative Correction and Eigenvector decomposition (ICE) (Imakaev et al., 2012b). Although these methods improve reproducibility of replicate Hi-C data (Imakaev et al., 2012a; Yaffe and Tanay, 2011), they do not explicitly account for the biases in *multiple* Hi-C data (Lun and Smyth, 2015; Stansfield et al., 2018b).

Critical Parameters

One of the main parameters when dealing with Hi-C data is the resolution of the data. At low resolutions, only large genomic changes can be detected. However, at low-resolution data are usually much more complete. Low-resolution data also help to mitigate issues due to processing time and memory constraints due to the much smaller matrix sizes. High-resolution Hi-C data can allow for insights into genomic looping such as what occurs in promoter-enhancer interactions. However, high-resolution data suffer from issues due to sparsity and analysis can be constrained by the computer hardware’s ability to handle large matrices. The protocols presented here provide three alternate methods for performing comparative analyses on Hi-C data. If the user has data already processed and stored in a contact matrix format, the steps shown in protocol 1 will likely be the easiest to perform. If the user is starting from raw data, protocol 2 is another viable option for analysis. If a high-resolution analysis is required, the steps presented in protocol 3 may be most appropriate. However, the long run times for FIND will need to be weighed against the benefits of using a spatially dependent model.

Troubleshooting

Analyses of Hi-C experiments can be complex and involve several different software packages. If the user obtains errors when attempting to apply the methods detailed in this unit a likely source is the data itself. Make sure that data are in the proper format for the software package being used. Additional problems could be due to data sparsity. If the Hi-C data were not sequenced at a deep enough level, the user might need to try a lower resolution to obtain meaningful results. Another possible problem that could occur when analyzing Hi-C data in the R environment is that memory requirements exceed the hardware capabilities, as R performs all operations in-memory. Hi-C data matrices are large, and analysis requires significant computing power. The user may have better results attempting to perform an analysis on a computing cluster if one is available. Other issues related specifically to certain software packages can be directed to the authors of the package either on GitHub or the Bioconductor support site.

Acknowledgments

This work was partially supported by the National Institute of Environmental Health Sciences of the National Institutes of Health [T32ES007334]. The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Institutes of Health.

Literature Cited

- Ay F, and Noble WS 2015 Analysis methods for studying the 3D architecture of the genome. *Genome Biol* 16:183. [PubMed: 26328929]
- Ballman KV OA, Grill DE 2004 Faster cyclic loess: Normalizing rna arrays via linear models. *Bioinformatics*.
- Barutcu AR, Lajoie BR, McCord RP, Tye CE, Hong D, Messier TL, Browne G, Wijnen AJ van, Lian JB, Stein JL, et al. 2015 Chromatin interaction analysis reveals changes in small chromosome and telomere clustering between epithelial and breast cancer cells. *Genome Biol* 16:214. [PubMed: 26415882]
- Bonev B, Mendelson Cohen N, Szabo Q, Fritsch L, Papadopoulos GL, Lubling Y, Xu X, Lv X, Hugnot J-P, Tanay A, et al. 2017 Multiscale 3D genome rewiring during mouse neural development. *Cell* 171:557–572.e24. [PubMed: 29053968]
- Dekker J ML, Marti-Renom MA 2013 Exploring the three-dimensional organization of genomes: Interpreting chromatin interaction data. *Nat Rev Genet*.
- Dekker J, Rippe K, Dekker M, and Kleckner N 2002 Capturing chromosome conformation. *Science* 295:1306–11. [PubMed: 11847345]
- Dixon JR, Jung I, Selvaraj S, Shen Y, Antosiewicz-Bourget JE, Lee AY, Ye Z, Kim A, Rajagopal N, Xie W, et al. 2015 Chromatin architecture reorganization during stem cell differentiation. *Nature* 518:331–6. [PubMed: 25693564]
- Dixon JR, Selvaraj S, Yue F, Kim A, Li Y, Shen Y, Hu M, Liu JS, and Ren B 2012 Topological domains in mammalian genomes identified by analysis of chromatin interactions. *Nature* 485:376–80. [PubMed: 22495300]
- Djekidel MN, Chen Y, and Zhang MQ 2018 FIND: DifFerential chromatin interactions detection using a spatial poisson process. *Genome Res*.
- Dozmorov MG, C. B. G, Lukas R. Cara 2016 GenomeRunner web server: Regulatory similarity and differences define the functional impact of snp sets. *Bioinformatics*.
- Dudoit S, Yang YH, Callow MJ, and Speed TP 2002 Statistical methods for identifying differentially expressed genes in replicated cDNA microarray experiments. *Statistica sinica*:111–139.
- Durand N, Shamim MS, and Aiden EL 2016 Juicer provides a one-click system for analyzing loop-resolution hi-c experiments. *Cell Systems* 3(1).
- Fisher R 1950 Statistical methods for research workers. Edinburgh: Oliver; BOyd.
- Fletez-Brant K, Qiu Y, Gorkin DU, Hu M, and Hansen KD 2017 Removing unwanted variation between samples in Hi-C experiments. –.
- Fudenberg G, Getz G, Meyerson M, and Mirny LA 2011 High order chromatin architecture shapes the landscape of chromosomal alterations in cancer. *Nat Biotechnol* 29:1109–13. [PubMed: 22101486]
- Imakaev M, Fudenberg G, McCord RP, Naumova N, Goloborodko A, Lajoie BR, Dekker J, and Mirny LA 2012a Iterative correction of hi-c data reveals hallmarks of chromosome organization. *Nat Methods* 9:999–1003. [PubMed: 22941365]
- Imakaev M, Fudenberg G, McCord RP, Naumova N, Goloborodko A, Lajoie BR, Dekker J, and Mirny LA 2012b Iterative correction of hi-c data reveals hallmarks of chromosome organization. *Nat Methods* 9:999–1003. [PubMed: 22941365]
- Karolchik ASH, Donna, and Kent WJ 2009 The ucsc genome browser. *Current protocols in bioinformatics*.
- Knight PA, and Ruiz D 2012 A fast algorithm for matrix balancing. *IMA Journal of Numerical Analysis*:drs019.
- Lajoie BR, Dekker J, Kaplan N 2015 The hitchhiker’s guide to hi-c analysis: Practical guidelines. *Methods*.
- Langmead S, B & Salzberg 2012 Fast gapped-read alignment with bowtie 2. *Nature Methods*.
- Lieberman-Aiden E, Berkum NL van, Williams L, Imakaev M, Ragoczy T, Telling A, Amit I, Lajoie BR, Sabo PJ, Dorschner MO, et al. 2009 Comprehensive mapping of long-range interactions reveals folding principles of the human genome. *Science* 326:289–93. [PubMed: 19815776]

- Lun ATL, and Smyth GK 2015 DiffHic: A bioconductor package to detect differential genomic interactions in hi-c data. *BMC Bioinformatics* 16:258. [PubMed: 26283514]
- Nguyen T, Tagett R, Donato M, Mitrea C, Draghici S 2016 A novel bi-level meta-analysis approach: Applied to biological pathway analysis. *Bioinformatics*, *Volume* 32, *Issue* 3.
- O'Sullivan JM, Hendy MD, Pichugina T, Wake GC, and Langowski J The statistical-mechanics of chromosome conformation capture. *Nucleus* 4:390–8. [PubMed: 24051548]
- Rao SSP, Huang S-C, Glenn St Hilaire B, Engreitz JM, Perez EM, Kieffer-Kwon K-R, Sanborn AL, Johnstone SE, Bascom GD, Bochkov ID, et al. 2017 Cohesin loss eliminates all loop domains. *Cell* 171:305–320.e24. Available at: 10.1016/j.cell.2017.09.026. [PubMed: 28985562]
- Rao SSP, Huntley MH, Durand NC, Stamenova EK, Bochkov ID, Robinson JT, Sanborn AL, Machol I, Omer AD, Lander ES, et al. 2014 A 3D map of the human genome at kilobase resolution reveals principles of chromatin looping. *Cell* 159:1665–80. [PubMed: 25497547]
- Richard Durbin HL & 2009 Fast and accurate short read alignment with burrows–Wheeler transform. *Bioinformatics*.
- Rickman DS, Soong TD, Moss B, Mosquera JM, Dlabal J, Terry S, MacDonald TY, Tripodi J, Bunting K, Najfeld V, et al. 2012 Oncogene-mediated alterations in chromatin conformation. *Proc Natl Acad Sci U S A* 109:9083–8. [PubMed: 22615383]
- Robinson M, and Smyth G 2008 Small sample estimation of negative binomial dispersion with application to sage data. *Biostatistics*.
- Servant N, B. R L, Varoquaux N, and Barillot E 2015 HiC-pro: An optimized and flexible pipeline for hi-c data processing. *Genome Biology* 16:259. [PubMed: 26619908]
- Sheffield Nathan C., and Bock C 2016 LOLA: Enrichment analysis for genomic region sets and regulatory elements in r and bioconductor. *Bioinformatics*.
- Stansfield JC, Cresswell KG, and Dozmorov MG 2018a MultiHiCcompare: Joint normalization and comparative analysis of complex hi-c experiments. Manuscript submitted.
- Stansfield JC, Cresswell KG, Vladimirov VI, and Dozmorov MG 2018b HiCcompare: An r-package for joint normalization and comparison of hi-c datasets. *BMC Bioinformatics* 19:279. [PubMed: 30064362]
- Stouffer ED, Suchman SA 1949 The american soldier, vol.1: Adjustment during army life. Princeton University Press.
- Taberlay PC, Achinger-Kawecka J, Lun ATL, Buske FA, Sabir K, Gould CM, Zotenko E, Bert SA, Giles KA, Bauer DC, et al. 2016 Three-dimensional disorganization of the cancer genome occurs coincident with long-range genetic and epigenetic alterations. *Genome Res* 26:719–31. [PubMed: 27053337]
- Voichita C, Donato M, and Draghici S 2012 Incorporating gene significance in the impact analysis of signaling pathways. *Proceedings of the International Conference on Machine Learning Applications (ICMLA)*.
- Yaffe E, and Tanay A 2011 Probabilistic modeling of hi-c contact maps eliminates systematic biases to characterize global chromosomal architecture. *Nat Genet* 43:1059–65. [PubMed: 22001755]

Significance Statement

This unit will introduce methods for the comparative (aka differential) analysis of the three-dimensional (3D) structure of the genome using data generated by high throughput chromatin conformation capture (Hi-C) technologies. Hi-C data allows for insights into the genome-wide genomic interactions which play an important role in the regulation of the genome. Just as differential expression analyses using RNA-seq data have become a routine part of genomic experiments, we expect the comparative analysis of genomic interactions to become a common task for a bioinformatician. This workflow will help a novice learn to perform a comparative analysis of two or more Hi-C datasets and interpret the results of the differential genomic interactions.

Figure 1. Workflow of a comparative Hi-C analysis using multiHiCcompare

Figure 2. MD plots of sample 3 vs. sample 4

A) before, and B) after normalization. The MD plots for the other pairs of samples, and on other chromosomes, look similar. X-axis is unit genomic distance which corresponds to each consecutive off-diagonal trace of the full contact matrix. Y-axis is the log2 difference between the two samples being compared. Red line represents the loess fit to the data. Any trends or shift away from $y = 0$ represent different between dataset biases which are ideally removed by the normalization procedure.

Figure 3. Composite MD plot with significant differentially interacting regions highlighted

Highlighted points display where the differential interactions are occurring in relation to unit genomic distance on the x-axis and log2 fold change on the y-axis. Points highlighted in yellow are moderately significant while points highlighted in red are highly significant.

Figure 4. Manhattan plot of region-summarized differential interaction statistics.

Y-axis shows A) addCLT-combined (Nguyen, 2016) p-values, B) counts of the number of times each region was found significant. These plots were created using all interactions. In part A) alternating black and gray coloring differentiate the points between chromosomes. Panels are meant to display the different options of representing the results in manhattan plot format.

Figure 5.

A) Manhattan plot of p-values from the counts object, combined using addCLT method. B) Manhattan plot of the counts from the counts object. These plots use only the filtered results of the topDirs function and thus are more significant than the interactions shown in figure 4.

Figure 6. MA plot generated by diffHiC pipeline

A) before, and B) after normalization. The x-axis represents the log₂ average IF value between the two datasets and the y-axis represents the log₂ difference between the two datasets. The red line represents the loess fit to the data. Normalization should ideally center the data around $y = 0$ and remove any trends between the datasets.

Figure 7. Diagnostic plots generated by diffHiC pipeline.

A) Biological coefficient of variation for each bin-pair against bin-pair abundance. B) Quasi-likelihood dispersion against bin-pair abundance. X-axes of both panels represent the average log counts per million (CPM), a measure of average IF value.

Figure 8. Interaction matrices of the most significantly different region.

A) Normal group. B) Auxin treated group. The rectangle represents the region that was detected as differential between the conditions.

Table 1.

Results of the ROntoTools pathways analysis. Columns indicate the ID of the KEGG canonical pathway, its description, the perturbation p-value, and the FDR adjusted perturbation p-value.

	pathNames	pPert	pPert.fdr
path:hsa04060	Cytokine-cytokine receptor interaction	0.005	0.099
path:hsa04080	Neuroactive ligand-receptor interaction	0.005	0.099
path:hsa04144	Endocytosis	0.005	0.099
path:hsa04145	Phagosome	0.005	0.099
path:hsa04146	Peroxisome	0.005	0.099
path:hsa04216	Ferroptosis	0.005	0.099
path:hsa04721	Synaptic vesicle cycle	0.005	0.099
path:hsa05166	Human T-cell leukemia virus 1 infection	0.005	0.099
path:hsa05220	Chronic myeloid leukemia	0.005	0.099
path:hsa05322	Systemic lupus erythematosus	0.005	0.099
path:hsa03460	Fanconi anemia pathway	0.010	0.099
path:hsa04010	MAPK signaling pathway	0.010	0.099
path:hsa04072	Phospholipase D signaling pathway	0.010	0.099
path:hsa04630	JAK-STAT signaling pathway	0.010	0.099
path:hsa04742	Taste transduction	0.010	0.099