

# Package ‘idr2d’

May 2, 2026

**Title** Irreproducible Discovery Rate for Genomic Interactions Data

**Version** 1.27.0

**Description** A tool to measure reproducibility between genomic experiments that produce two-dimensional peaks (interactions between peaks), such as ChIA-PET, HiChIP, and HiC. idr2d is an extension of the original idr package, which is intended for (one-dimensional) ChIP-seq peaks.

**License** MIT + file LICENSE

**URL** <https://idr2d.mit.edu>

**Depends** R (>= 3.6)

**Imports** dplyr (>= 0.7.6), futile.logger (>= 1.4.3), GenomeInfoDb (>= 1.14.0), GenomicRanges (>= 1.30), ggplot2 (>= 3.1.1), grDevices, grid, idr (>= 1.2), IRanges (>= 2.18.0), magrittr (>= 1.5), methods, reticulate (>= 1.13), scales (>= 1.0.0), stats, stringr (>= 1.3.1), utils

**Suggests** DT (>= 0.4), htmltools (>= 0.3.6), knitr (>= 1.20), rmarkdown (>= 1.10), roxygen2 (>= 6.1.0), testthat (>= 2.1.0)

**VignetteBuilder** knitr

**biocViews** DNA3DStructure, GeneRegulation, PeakDetection, Epigenetics, FunctionalGenomics, Classification, HiC

**Encoding** UTF-8

**RoxygenNote** 7.3.3

**SystemRequirements** Python (>= 3.5.0), hic-straw

**git\_url** <https://git.bioconductor.org/packages/idr2d>

**git\_branch** devel

**git\_last\_commit** 798ff6c

**git\_last\_commit\_date** 2026-04-28

**Repository** Bioconductor 3.24

**Date/Publication** 2026-05-01

**Author** Konstantin Krismer [aut, cre, cph] (ORCID:  
<<https://orcid.org/0000-0001-8994-3416>>),  
David Gifford [ths, cph] (ORCID:  
<<https://orcid.org/0000-0003-1709-4034>>)

**Maintainer** Konstantin Krismer <krismer@mit.edu>

## Contents

calculate_midpoint_distance1d . . . . .	2
calculate_midpoint_distance2d . . . . .	3
calculate_relative_overlap1d . . . . .	5
calculate_relative_overlap2d . . . . .	6
chiapet . . . . .	7
chipseq . . . . .	8
determine_anchor_overlap . . . . .	8
draw_hic_contact_map . . . . .	9
draw_idr_distribution_histogram . . . . .	10
draw_rank_idr_scatterplot . . . . .	11
draw_value_idr_scatterplot . . . . .	12
establish_bijection . . . . .	13
establish_bijection1d . . . . .	14
establish_bijection2d . . . . .	16
establish_overlap1d . . . . .	17
establish_overlap2d . . . . .	19
estimate_idr . . . . .	21
estimate_idr1d . . . . .	22
estimate_idr2d . . . . .	25
estimate_idr2d_hic . . . . .	27
hic . . . . .	29
parse_hic_pro_matrix . . . . .	30
parse_juicer_matrix . . . . .	30
preprocess . . . . .	31
remove_nonstandard_chromosomes1d . . . . .	33
remove_nonstandard_chromosomes2d . . . . .	33
<b>Index</b>	<b>35</b>

---

calculate\_midpoint\_distance1d

*Distance between Midpoints of two Peaks*

---

### Description

Calculates the distance in nucleotides between the midpoints of two peaks.

Note: peaks must be on the same chromosome; start coordinate is always less than end coordinate

### Usage

calculate\_midpoint\_distance1d(peak1\_start, peak1\_end, peak2\_start, peak2\_end)

### Arguments

peak1_start	integer vector; genomic start coordinate(s) of peak in replicate 1
peak1_end	integer vector; genomic end coordinate(s) of peak in replicate 1
peak2_start	integer vector; genomic start coordinate(s) of peak in replicate 2
peak2_end	integer vector; genomic end coordinate(s) of peak in replicate 2

**Value**

positive integer vector; distances between peak pairs

**Examples**

```
# identical, zero distance
calculate_midpoint_distance1d(100, 120,
                             100, 120)

# centered, zero distance
calculate_midpoint_distance1d(100, 120,
                             90, 130)

# off by 10 per anchor
calculate_midpoint_distance1d(100, 120,
                             110, 130)

# vectorized example
calculate_midpoint_distance1d(c(100, 100, 100),
                             c(120, 120, 120),
                             c(100, 90, 110),
                             c(120, 130, 130))
```

---

calculate\_midpoint\_distance2d

*Distance between Anchor Midpoints of two Interactions*

---

**Description**

Calculates the distance in nucleotides between the anchor midpoints of two interactions, which is the sum of the distance between midpoints of anchor A in interaction 1 and anchor A in interaction 2, and the distance between midpoints of anchor B in interaction 1 and anchor B in interaction 2.

Note: all anchors must be on the same chromosome; start coordinate is always less than end coordinate

**Usage**

```
calculate_midpoint_distance2d(
  int1_anchor_a_start,
  int1_anchor_a_end,
  int1_anchor_b_start,
  int1_anchor_b_end,
  int2_anchor_a_start,
  int2_anchor_a_end,
  int2_anchor_b_start,
  int2_anchor_b_end
)
```

**Arguments**

`int1_anchor_a_start`  
integer vector; genomic start coordinate(s) of anchor A in replicate 1 interaction

`int1_anchor_a_end`  
integer vector; genomic end coordinate(s) of anchor A in replicate 1 interaction

`int1_anchor_b_start`  
integer vector; genomic start coordinate(s) of anchor B in replicate 1 interaction

`int1_anchor_b_end`  
integer vector; genomic end coordinate(s) of anchor B in replicate 1 interaction

`int2_anchor_a_start`  
integer vector; genomic start coordinate(s) of anchor A in replicate 2 interaction

`int2_anchor_a_end`  
integer vector; genomic end coordinate(s) of anchor A in replicate 2 interaction

`int2_anchor_b_start`  
integer vector; genomic start coordinate(s) of anchor B in replicate 2 interaction

`int2_anchor_b_end`  
integer vector; genomic end coordinate(s) of anchor B in replicate 2 interaction

**Value**

positive integer vector; distances between interaction pairs

**Examples**

```
# identical, zero distance
calculate_midpoint_distance2d(100, 120, 240, 260,
                             100, 120, 240, 260)

# centered, zero distance
calculate_midpoint_distance2d(100, 120, 240, 260,
                             90, 130, 230, 270)

# off by 10 per anchor
calculate_midpoint_distance2d(100, 120, 240, 250,
                             110, 130, 230, 240)

# off by 10 (anchor B only)
calculate_midpoint_distance2d(100, 120, 240, 250,
                             90, 130, 250, 260)

# vectorized example
calculate_midpoint_distance2d(c(100, 100, 100, 100),
                             c(120, 120, 120, 120),
                             c(240, 240, 240, 240),
                             c(260, 260, 250, 250),
                             c(100, 90, 110, 90),
                             c(120, 130, 130, 130),
                             c(240, 230, 230, 250),
                             c(260, 270, 240, 260))
```

---

`calculate_relative_overlap1d`*Relative Anchor Overlap of two Peaks*

---

**Description**

Calculates the overlap between anchor A of interaction 1 and anchor A of interaction 2, as well as anchor B of interaction 1 and anchor B of interaction 2. The overlap (in nucleotides) is then normalized by the length of the anchors.

**Usage**

```
calculate_relative_overlap1d(peak1_start, peak1_end, peak2_start, peak2_end)
```

**Arguments**

peak1_start	integer vector; genomic start coordinate(s) of peak in replicate 1
peak1_end	integer vector; genomic end coordinate(s) of peak in replicate 1
peak2_start	integer vector; genomic start coordinate(s) of peak in replicate 2
peak2_end	integer vector; genomic end coordinate(s) of peak in replicate 2

**Value**

numeric vector; relative overlaps between peak pairs

**Examples**

```
# 100% overlap
calculate_relative_overlap1d(100, 120,
                           100, 120)

# 50% overlap
calculate_relative_overlap1d(100, 120,
                           100, 110)

# negative overlap
calculate_relative_overlap1d(100, 120,
                           130, 140)

# larger negative overlap
calculate_relative_overlap1d(100, 120,
                           200, 220)

# vectorized example
calculate_relative_overlap1d(c(100, 100, 100, 100),
                           c(120, 120, 120, 120),
                           c(100, 100, 130, 200),
                           c(120, 110, 140, 220))
```

---

 calculate\_relative\_overlap2d

*Relative Anchor Overlap of two Interactions*


---

### Description

Calculates the overlap between anchor A of interaction 1 and anchor A of interaction 2, as well as anchor B of interaction 1 and anchor B of interaction 2. The overlap (in nucleotides) is then normalized by the length of the anchors.

Note: anchors A and B of the same interaction have to be on the same chromosome; start coordinate is always less than end coordinate

### Usage

```
calculate_relative_overlap2d(
  int1_anchor_a_start,
  int1_anchor_a_end,
  int1_anchor_b_start,
  int1_anchor_b_end,
  int2_anchor_a_start,
  int2_anchor_a_end,
  int2_anchor_b_start,
  int2_anchor_b_end
)
```

### Arguments

```
int1_anchor_a_start
    integer vector; genomic start coordinate(s) of anchor A in replicate 1 interaction
int1_anchor_a_end
    integer vector; genomic end coordinate(s) of anchor A in replicate 1 interaction
int1_anchor_b_start
    integer vector; genomic start coordinate(s) of anchor B in replicate 1 interaction
int1_anchor_b_end
    integer vector; genomic end coordinate(s) of anchor B in replicate 1 interaction
int2_anchor_a_start
    integer vector; genomic start coordinate(s) of anchor A in replicate 2 interaction
int2_anchor_a_end
    integer vector; genomic end coordinate(s) of anchor A in replicate 2 interaction
int2_anchor_b_start
    integer vector; genomic start coordinate(s) of anchor B in replicate 2 interaction
int2_anchor_b_end
    integer vector; genomic end coordinate(s) of anchor B in replicate 2 interaction
```

### Value

numeric vector; relative overlaps between interaction pairs

**Examples**

```

# 100% overlap
calculate_relative_overlap2d(100, 120, 240, 260,
                             100, 120, 240, 260)

# 50% overlap
calculate_relative_overlap2d(100, 120, 240, 250,
                             100, 110, 240, 260)

# negative overlap
calculate_relative_overlap2d(100, 120, 240, 250,
                             130, 140, 260, 280)

# larger negative overlap
calculate_relative_overlap2d(100, 120, 240, 250,
                             200, 220, 340, 350)

# vectorized example
calculate_relative_overlap2d(c(100, 100, 100, 100),
                             c(120, 120, 120, 120),
                             c(240, 240, 240, 240),
                             c(260, 250, 250, 250),
                             c(100, 100, 130, 200),
                             c(120, 110, 140, 220),
                             c(240, 240, 260, 340),
                             c(260, 260, 280, 350))

```

---

chiapet

*Example Genomic Interaction Data Set*


---

**Description**

This object contains genomic interactions on chromosomes 1 to 5, which could be the results of Hi-C or ChIA-PET experiments, done in duplicates.

**Usage**

```
chiapet
```

**Format**

A list with two components, the data frames rep1\_df and rep2\_df, which have the following seven columns:

column 1:	chr_a	character; genomic location of anchor A - chromosome (e.g., "chr3")
column 2:	start_a	integer; genomic location of anchor A - start coordinate
column 3:	end_a	integer; genomic location of anchor A - end coordinate
column 4:	chr_b	character; genomic location of anchor B - chromosome (e.g., "chr3")
column 5:	start_b	integer; genomic location of anchor B - start coordinate
column 6:	end_b	integer; genomic location of anchor B - end coordinate
column 7:	fdr	numeric; False Discovery Rate - significance of interaction

---

 chipseq

*Example Genomic Peak Data Set*


---

### Description

This object contains genomic peaks from two replicate ChIP-seq experiments.

### Usage

```
chipseq
```

### Format

A list with two components, the data frames rep1\_df and rep2\_df, which have the following four columns:

column 1:	chr	character; genomic location of peak - chromosome (e.g., "chr3")
column 2:	start	integer; genomic location of peak - start coordinate
column 3:	end	integer; genomic location of peak - end coordinate
column 4:	value	numeric; heuristic used to rank the peaks

---

 determine\_anchor\_overlap

*Identifies Overlapping Anchors*


---

### Description

Identifies all overlapping anchor pairs (m:n mapping).

### Usage

```
determine_anchor_overlap(rep1_anchor, rep2_anchor, max_gap = -1L)
```

### Arguments

rep1\_anchor      data frame with the following columns:

column 1:	chr	character; genomic location of anchor in replicate 1 - chromosome (e.g., "chr3")
column 2:	start	integer; genomic location of anchor in replicate 1 - start coordinate
column 3:	end	integer; genomic location of anchor in replicate 1 - end coordinate

rep2\_anchor      data frame with the following columns:

column 1:	chr	character; genomic location of anchor in replicate 2 - chromosome (e.g., "chr3")
column 2:	start	integer; genomic location of anchor in replicate 2 - start coordinate
column 3:	end	integer; genomic location of anchor in replicate 2 - end coordinate

max\_gap          integer; maximum gap in nucleotides allowed between two anchors for them to be considered as overlapping (defaults to -1, i.e., overlapping anchors)

**Value**

A data frame containing overlapping anchor pairs with the following columns:

```
column 1:  rep1_idx  anchor index in data frame rep1_anchor
column 2:  rep2_idx  anchor index in data frame rep2_anchor
```

**Examples**

```
rep1_df <- idr2d::chiapet$rep1_df
rep2_df <- idr2d::chiapet$rep2_df

rep1_anchor_a <- data.frame(chr = rep1_df[, 1],
                           start = rep1_df[, 2],
                           end = rep1_df[, 3])
rep2_anchor_a <- data.frame(chr = rep2_df[, 1],
                           start = rep2_df[, 2],
                           end = rep2_df[, 3])

anchor_a_overlap <- determine_anchor_overlap(rep1_anchor_a, rep2_anchor_a)
```

---

draw\_hic\_contact\_map *Create Hi-C contact map*

---

**Description**

Creates Hi-C contact maps to visualize the results of [estimate\\_idr2d\\_hic](#).

**Usage**

```
draw_hic_contact_map(
  df,
  idr_cutoff = NULL,
  chromosome = NULL,
  start_coordinate = NULL,
  end_coordinate = NULL,
  title = NULL,
  values_normalized = FALSE,
  log_values = TRUE
)
```

**Arguments**

df                    output of [estimate\\_idr2d\\_hic](#), a data frame with the following columns:

column 1:	interaction	character; genomic location of interaction block (e.g., "chr1:204940000-204940000")
column 2:	value	numeric; p-value, FDR, or heuristic used to rank the interactions
column 3:	"rep_value"	numeric; value of corresponding replicate interaction
column 4:	"rank"	integer; rank of the interaction, established by value column, ascending order
column 5:	"rep_rank"	integer; rank of corresponding replicate interaction
column 6:	"idr"	integer; IDR of the block and the corresponding block in the other replicate

<code>idr_cutoff</code>	numeric; only show blocks with $IDR < idr\_cutoff$ , shows all blocks by default
<code>chromosome</code>	character; chromosome name or list of chromosome names to be analyzed, e.g., UCSC chromosome 1, "chr1", defaults to all chromosomes ( <code>chromosome = NULL</code> )
<code>start_coordinate</code>	integer; only show contact map window between "start_coordinate" and "end_coordinate", by default shows entire chromosome
<code>end_coordinate</code>	integer; only show contact map window between "start_coordinate" and "end_coordinate", by default shows entire chromosome
<code>title</code>	character; plot title
<code>values_normalized</code>	logical; are read counts in value column raw or normalized? Defaults to FALSE
<code>log_values</code>	logical; log-transform value column? Defaults to TRUE

**Value**

ggplot2 object; Hi-C contact map

**Examples**

```
idr_results_df <- estimate_idr2d_hic(idr2d:::hic$rep1_df,
                                   idr2d:::hic$rep2_df)
draw_hic_contact_map(idr_results_df, idr_cutoff = 0.05, chromosome = "chr1")
```

---

`draw_idr_distribution_histogram`  
*Create histogram of IDR values*

---

**Description**

Creates diagnostic plots to visualize the results of `estimate_idr`.

**Usage**

```
draw_idr_distribution_histogram(
  df,
  remove_na = TRUE,
  xlab = "IDR",
  ylab = "density",
  title = "IDR value distribution"
)
```

**Arguments**

`df` part of output of `estimate_idr`, a data frame with at least the following named columns:

`idr` IDR of the peak and the corresponding peak in the other replicate.

remove_na	logical; should NA values be removed?
xlab	character; x axis label
ylab	character; y axis label
title	character; plot title

**Value**

ggplot2 object; IDR distribution histogram

**Examples**

```
idr_results <- estimate_idr1d(idr2d:::chipseq$rep1_df,
                             idr2d:::chipseq$rep2_df,
                             value_transformation = "log")
draw_idr_distribution_histogram(idr_results$rep1_df)
```

---

draw\_rank\_idr\_scatterplot

*Create scatterplot of IDR values*

---

**Description**

Creates diagnostic plots to visualize the results of [estimate\\_idr](#).

**Usage**

```
draw_rank_idr_scatterplot(
  df,
  remove_na = TRUE,
  xlab = "rank in replicate 1",
  ylab = "rank in replicate 2",
  log_idr = FALSE,
  title = "rank - IDR dependence",
  color_gradient = c("rainbow", "default"),
  alpha = 1,
  max_points_shown = 2500
)
```

**Arguments**

df	part of output of <a href="#">estimate_idr</a> , a data frame with at least the following named columns:
rank	integer; rank of the peak, established by value column, ascending order
rep_rank	integer; rank of corresponding replicate peak.
idr	IDR of the peak and the corresponding peak in the other replicate.
remove_na	logical; should NA values be removed?
xlab	character; x axis label

ylab	character; y axis label
log_idr	logical; use logarithmized IDRs for colors to better distinguish highly significant IDRs
title	character; plot title
color_gradient	character; either "rainbow" or "default"
alpha	numeric; transparency of dots, from 0.0 - 1.0, where 1.0 is completely opaque; default is 1.0
max_points_shown	integer; default is 2500

**Value**

ggplot2 object; IDR rank scatterplot

**Examples**

```
idr_results <- estimate_idr1d(idr2d:::chipseq$rep1_df,
                             idr2d:::chipseq$rep2_df,
                             value_transformation = "log")
draw_rank_idr_scatterplot(idr_results$rep1_df)
```

---

draw\_value\_idr\_scatterplot

*Create scatterplot of IDR values*

---

**Description**

Creates diagnostic plots to visualize the results of `estimate_idr`.

**Usage**

```
draw_value_idr_scatterplot(
  df,
  remove_na = TRUE,
  remove_outliers = TRUE,
  xlab = "transformed value in replicate 1",
  ylab = "transformed value in replicate 2",
  log_axes = FALSE,
  log_idr = FALSE,
  title = "value - IDR dependence",
  color_gradient = c("rainbow", "default"),
  alpha = 1,
  max_points_shown = 2500
)
```

**Arguments**

df	part of output of <code>estimate_idr</code> , a data frame with at least the following named columns:
value	numeric; p-value, FDR, or heuristic used to rank the peaks
rep_value	numeric; value of corresponding replicate peak
idr	IDR of the peak and the corresponding peak in the other replicate.
remove_na	logical; should NA values be removed?
remove_outliers	logical; removes extreme data points
xlab	character; x axis label
ylab	character; y axis label
log_axes	logical; show logarithmized values from replicate 1 and 2 (default value is FALSE)
log_idr	logical; use logarithmized IDRs for colors to better distinguish highly significant IDRs (default value is FALSE)
title	character; plot title
color_gradient	character; either "rainbow" or "default"
alpha	numeric; transparency of dots, from 0.0 - 1.0, where 1.0 is completely opaque; default is 1.0
max_points_shown	integer; default is 2500

**Value**

ggplot2 object; IDR value scatterplot

**Examples**

```
idr_results <- estimate_idr1d(idr2d::chipseq$rep1_df,
                             idr2d::chipseq$rep2_df,
                             value_transformation = "log")
draw_value_idr_scatterplot(idr_results$rep1_df)
```

---

establish\_bijection     *Finds One-to-One Correspondence between Peaks or interactions from Replicate 1 and 2*

---

**Description**

This method establishes a bijective assignment between observations (genomic peaks in case of ChIP-seq, genomic interactions in case of ChIA-PET, HiChIP, and Hi-C) from replicate 1 and 2. An observation in replicate 1 is assigned to an observation in replicate 2 if and only if (1) the observation loci in both replicates overlap (or the gap between them is less than or equal to `max_gap`), and (2) there is no other observation in replicate 2 that overlaps with the observation in replicate 1 and has a lower *ambiguity resolution value*.

**Usage**

```

establish_bijection(
  rep1_df,
  rep2_df,
  analysis_type = c("IDR1D", "IDR2D"),
  ambiguity_resolution_method = c("overlap", "midpoint", "value"),
  max_gap = -1L
)

```

**Arguments**

rep1_df	data frame of observations (i.e., genomic peaks or genomic interactions) of replicate 1. If analysis_type is IDR1D, the columns of rep1_df are described in <a href="#">establish_bijection1d</a> , otherwise in <a href="#">establish_bijection2d</a>
rep2_df	data frame of observations (i.e., genomic peaks or genomic interactions) of replicate 2. Same columns as rep1_df.
analysis_type	"IDR2D" for genomic interaction data sets, "IDR1D" for genomic peak data sets
ambiguity_resolution_method	defines how ambiguous assignments (when one interaction or peak in replicate 1 overlaps with multiple interactions or peaks in replicate 2 or vice versa) are resolved. For available methods, see <a href="#">establish_overlap1d</a> or <a href="#">establish_overlap2d</a> , respectively.
max_gap	integer; maximum gap in nucleotides allowed between two anchors for them to be considered as overlapping (defaults to -1, i.e., overlapping anchors)

**Value**

See [establish\\_bijection1d](#) or [establish\\_bijection2d](#), respectively.

**Examples**

```

rep1_df <- idr2d::chipseq$rep1_df
rep1_df$value <- preprocess(rep1_df$value, "log")

rep2_df <- idr2d::chipseq$rep2_df
rep2_df$value <- preprocess(rep2_df$value, "log")

mapping <- establish_bijection(rep1_df, rep2_df, analysis_type = "IDR1D")

```

---

establish\_bijection1d *Finds One-to-One Correspondence between Peaks from Replicate 1 and 2*

---

**Description**

This method establishes a bijective assignment between peaks from replicate 1 and 2. A peak in replicate 1 is assigned to a peak in replicate 2 if and only if (1) they overlap (or the gap between the peaks is less than or equal to max\_gap), and (2) there is no other peak in replicate 2 that overlaps with the peak in replicate 1 and has a lower *ambiguity resolution value*.

**Usage**

```

establish_bijection1d(
  rep1_df,
  rep2_df,
  ambiguity_resolution_method = c("overlap", "midpoint", "value"),
  max_gap = -1L
)

```

**Arguments**

`rep1_df` data frame of observations (i.e., genomic peaks) of replicate 1, with at least the following columns (position of columns matter, column names are irrelevant):

column 1:	<code>chr</code>	character; genomic location of peak - chromosome (e.g., "chr3")
column 2:	<code>start</code>	integer; genomic location of peak - start coordinate
column 3:	<code>end</code>	integer; genomic location of peak - end coordinate
column 4:	<code>value</code>	numeric; p-value, FDR, or heuristic used to rank the interactions

`rep2_df` data frame of observations (i.e., genomic peaks) of replicate 2, with the following columns (position of columns matter, column names are irrelevant):

column 1:	<code>chr</code>	character; genomic location of peak - chromosome (e.g., "chr3")
column 2:	<code>start</code>	integer; genomic location of peak - start coordinate
column 3:	<code>end</code>	integer; genomic location of peak - end coordinate
column 4:	<code>value</code>	numeric; p-value, FDR, or heuristic used to rank the interactions

`ambiguity_resolution_method`

defines how ambiguous assignments (when one interaction in replicate 1 overlaps with multiple interactions in replicate 2 or vice versa) are resolved. Available methods:

"value"	interactions are prioritized by ascending or descending value column (see <code>sorting_direction</code> ), e.g., if two interactions overlap, the one with the higher value is chosen.
"overlap"	the interaction pair is chosen which has the highest relative overlap, i.e., overlap in nucleotides of replicate 1 divided by the sum of lengths of the two interactions.
"midpoint"	the interaction pair is chosen which has the smallest distance between their anchor midpoints, i.e., distance between the midpoint of the two interactions.

`max_gap` integer; maximum gap in nucleotides allowed between two anchors for them to be considered as overlapping (defaults to -1, i.e., overlapping anchors)

**Value**

Data frames `rep1_df` and `rep2_df` with the following columns:

column 1:	<code>chr</code>	character; genomic location of peak - chromosome (e.g., "chr3")
column 2:	<code>start</code>	integer; genomic location of peak - start coordinate
column 3:	<code>end</code>	integer; genomic location of peak - end coordinate
column 4:	<code>value</code>	numeric; p-value, FDR, or heuristic used to rank the peaks
column 5:	<code>rep_value</code>	numeric; value of corresponding replicate peak. If no corresponding peak was found, <code>rep_value</code> is NA.
column 6:	<code>rank</code>	integer; rank of the peak, established by value column, ascending order
column 7:	<code>rep_rank</code>	integer; rank of corresponding replicate peak. If no corresponding peak was found, <code>rep_rank</code> is NA.
column 8:	<code>idx</code>	integer; peak index, primary key
column 9:	<code>rep_idx</code>	integer; specifies the index of the corresponding peak in the other replicate (foreign key). If no corresponding peak was found, <code>rep_idx</code> is NA.

**Examples**

```

rep1_df <- idr2d::chipseq$rep1_df
rep1_df$value <- preprocess(rep1_df$value, "log")

rep2_df <- idr2d::chipseq$rep2_df
rep2_df$value <- preprocess(rep2_df$value, "log")

mapping <- establish_bijection1d(rep1_df, rep2_df)

```

---

establish\_bijection2d *Finds One-to-One Correspondence between Interactions from Replicate 1 and 2*

---

**Description**

This method establishes a bijective assignment between interactions from replicate 1 and 2. An interaction in replicate 1 is assigned to an interaction in replicate 2 if and only if (1) both anchors of the interactions overlap (or the gap between anchor A/B in replicate 1 and 2 is less than or equal to `max_gap`), and (2) there is no other interaction in replicate 2 that overlaps with the interaction in replicate 1 and has a lower *ambiguity resolution value*.

**Usage**

```

establish_bijection2d(
  rep1_df,
  rep2_df,
  ambiguity_resolution_method = c("overlap", "midpoint", "value"),
  max_gap = -1L
)

```

**Arguments**

`rep1_df` data frame of observations (i.e., genomic interactions) of replicate 1, with at least the following columns (position of columns matter, column names are irrelevant):

column 1:	<code>chr_a</code>	character; genomic location of anchor A - chromosome (e.g., "chr3")
column 2:	<code>start_a</code>	integer; genomic location of anchor A - start coordinate
column 3:	<code>end_a</code>	integer; genomic location of anchor A - end coordinate
column 4:	<code>chr_b</code>	character; genomic location of anchor B - chromosome (e.g., "chr3")
column 5:	<code>start_b</code>	integer; genomic location of anchor B - start coordinate
column 6:	<code>end_b</code>	integer; genomic location of anchor B - end coordinate
column 7:	<code>value</code>	numeric; p-value, FDR, or heuristic used to rank the interactions

`rep2_df` data frame of observations (i.e., genomic interactions) of replicate 2, with the following columns (position of columns matter, column names are irrelevant):

column 1:	<code>chr_a</code>	character; genomic location of anchor A - chromosome (e.g., "chr3")
column 2:	<code>start_a</code>	integer; genomic location of anchor A - start coordinate
column 3:	<code>end_a</code>	integer; genomic location of anchor A - end coordinate

column 4:	chr_b	character; genomic location of anchor B - chromosome (e.g., "chr3")
column 5:	start_b	integer; genomic location of anchor B - start coordinate
column 6:	end_b	integer; genomic location of anchor B - end coordinate
column 7:	value	numeric; p-value, FDR, or heuristic used to rank the interactions

#### ambiguity\_resolution\_method

defines how ambiguous assignments (when one interaction in replicate 1 overlaps with multiple interactions in replicate 2 or vice versa) are resolved. Available methods:

"value"	interactions are prioritized by ascending or descending value column (see <code>sorting_direction</code> ), e.g., if t
"overlap"	the interaction pair is chosen which has the highest relative overlap, i.e., overlap in nucleotides of replicate
"midpoint"	the interaction pair is chosen which has the smallest distance between their anchor midpoints, i.e., distance

max_gap	integer; maximum gap in nucleotides allowed between two anchors for them to be considered as overlapping (defaults to -1, i.e., overlapping anchors)
---------	--

### Value

Data frames `rep1_df` and `rep2_df` with the following columns:

column 1:	chr_a	character; genomic location of anchor A - chromosome (e.g., "chr3")
column 2:	start_a	integer; genomic location of anchor A - start coordinate
column 3:	end_a	integer; genomic location of anchor A - end coordinate
column 4:	chr_b	character; genomic location of anchor B - chromosome (e.g., "chr3")
column 5:	start_b	integer; genomic location of anchor B - start coordinate
column 6:	end_b	integer; genomic location of anchor B - end coordinate
column 7:	value	numeric; p-value, FDR, or heuristic used to rank the interactions
column 8:	"rep_value"	numeric; value of corresponding replicate interaction. If no corresponding interaction was f
column 9:	"rank"	integer; rank of the interaction, established by value column, ascending order
column 10:	"rep_rank"	integer; rank of corresponding replicate interaction. If no corresponding interaction was fo
column 11:	"idx"	integer; interaction index, primary key
column 12:	"rep_idx"	integer; specifies the index of the corresponding interaction in the other replicate (foreign k

### Examples

```
rep1_df <- idr2d::chiapet$rep1_df
rep1_df$fdr <- preprocess(rep1_df$fdr, "log_additive_inverse")

rep2_df <- idr2d::chiapet$rep2_df
rep2_df$fdr <- preprocess(rep2_df$fdr, "log_additive_inverse")

mapping <- establish_bijection2d(rep1_df, rep2_df)
```

## Description

This method returns all overlapping interactions between two replicates. For each pair of overlapping interactions, the *ambiguity resolution value* (ARV) is calculated, which helps to reduce the m:n mapping to a 1:1 mapping. The semantics of the ARV depend on the specified `ambiguity_resolution_method`, but in general interaction pairs with lower ARVs have priority over interaction pairs with higher ARVs when the bijective mapping is established.

## Usage

```
establish_overlap1d(
  rep1_df,
  rep2_df,
  ambiguity_resolution_method = c("overlap", "midpoint", "value"),
  max_gap = -1L
)
```

## Arguments

- `rep1_df` data frame of observations (i.e., genomic peaks) of replicate 1, with at least the following columns (position of columns matter, column names are irrelevant):
- |           |                    |   |
|-----------|--------------------|---|
| column 1: | <code>chr</code>   | character; genomic location of peak - chromosome (e.g., "chr3")   |
| column 2: | <code>start</code> | integer; genomic location of peak - start coordinate              |
| column 3: | <code>end</code>   | integer; genomic location of peak - end coordinate                |
| column 4: | <code>value</code> | numeric; p-value, FDR, or heuristic used to rank the interactions |
- `rep2_df` data frame of observations (i.e., genomic peaks) of replicate 2, with the following columns (position of columns matter, column names are irrelevant):
- |           |                    |   |
|-----------|--------------------|---|
| column 1: | <code>chr</code>   | character; genomic location of peak - chromosome (e.g., "chr3")   |
| column 2: | <code>start</code> | integer; genomic location of peak - start coordinate              |
| column 3: | <code>end</code>   | integer; genomic location of peak - end coordinate                |
| column 4: | <code>value</code> | numeric; p-value, FDR, or heuristic used to rank the interactions |
- `ambiguity_resolution_method` defines how ambiguous assignments (when one interaction in replicate 1 overlaps with multiple interactions in replicate 2 or vice versa) are resolved. Available methods:
- |            |  |
|------------|--|
| "value"    | interactions are prioritized by ascending or descending value column (see <code>sorting_direction</code> ), e.g., if t |
| "overlap"  | the interaction pair is chosen which has the highest relative overlap, i.e., overlap in nucleotides of replicate       |
| "midpoint" | the interaction pair is chosen which has the smallest distance between their anchor midpoints, i.e., distance          |
- `max_gap` integer; maximum gap in nucleotides allowed between two anchors for them to be considered as overlapping (defaults to -1, i.e., overlapping anchors)

## Value

data frame with the following columns:

- |           |                       |  |
|-----------|-----------------------|--|
| column 1: | <code>rep1_idx</code> | index of interaction in replicate 1 (i.e., row index in <code>rep1_df</code> ) |
| column 2: | <code>rep2_idx</code> | index of interaction in replicate 2 (i.e., row index in <code>rep2_df</code> ) |

column 3: arv            ambiguity resolution value used turn m:n mapping into 1:1 mapping. Interaction pairs with lower

### Examples

```
rep1_df <- idr2d::chipseq$rep1_df
rep1_df$value <- preprocess(rep1_df$value, "log_additive_inverse")

rep2_df <- idr2d::chipseq$rep2_df
rep2_df$value <- preprocess(rep2_df$value, "log_additive_inverse")

# shuffle to break preexisting order
rep1_df <- rep1_df[sample.int(nrow(rep1_df)), ]
rep2_df <- rep2_df[sample.int(nrow(rep2_df)), ]

# sort by value column
rep1_df <- dplyr::arrange(rep1_df, value)
rep2_df <- dplyr::arrange(rep2_df, value)

pairs_df <- establish_overlap1d(rep1_df, rep2_df)
```

---

establish\_overlap2d    *Establish m:n mapping between interactions from replicate 1 and 2*

---

### Description

This method returns all overlapping interactions between two replicates. For each pair of overlapping interactions, the *ambiguity resolution value* (ARV) is calculated, which helps to reduce the m:n mapping to a 1:1 mapping. The semantics of the ARV depend on the specified `ambiguity_resolution_method`, but in general interaction pairs with lower ARVs have priority over interaction pairs with higher ARVs when the bijective mapping is established.

### Usage

```
establish_overlap2d(
  rep1_df,
  rep2_df,
  ambiguity_resolution_method = c("overlap", "midpoint", "value"),
  max_gap = -1L
)
```

### Arguments

`rep1_df`            data frame of observations (i.e., genomic interactions) of replicate 1, with at least the following columns (position of columns matter, column names are irrelevant):

column 1:	<code>chr_a</code>	character; genomic location of anchor A - chromosome (e.g., "chr3")
column 2:	<code>start_a</code>	integer; genomic location of anchor A - start coordinate
column 3:	<code>end_a</code>	integer; genomic location of anchor A - end coordinate
column 4:	<code>chr_b</code>	character; genomic location of anchor B - chromosome (e.g., "chr3")
column 5:	<code>start_b</code>	integer; genomic location of anchor B - start coordinate

column 6: end\_b integer; genomic location of anchor B - end coordinate  
 column 7: value numeric; p-value, FDR, or heuristic used to rank the interactions

rep2\_df data frame of observations (i.e., genomic interactions) of replicate 2, with the following columns (position of columns matter, column names are irrelevant):

column 1: chr\_a character; genomic location of anchor A - chromosome (e.g., "chr3")  
 column 2: start\_a integer; genomic location of anchor A - start coordinate  
 column 3: end\_a integer; genomic location of anchor A - end coordinate  
 column 4: chr\_b character; genomic location of anchor B - chromosome (e.g., "chr3")  
 column 5: start\_b integer; genomic location of anchor B - start coordinate  
 column 6: end\_b integer; genomic location of anchor B - end coordinate  
 column 7: value numeric; p-value, FDR, or heuristic used to rank the interactions

ambiguity\_resolution\_method

defines how ambiguous assignments (when one interaction in replicate 1 overlaps with multiple interactions in replicate 2 or vice versa) are resolved. Available methods:

"value" interactions are prioritized by ascending or descending value column (see `sorting_direction`), e.g., if two interactions have the same value, the interaction pair is chosen which has the highest relative overlap, i.e., overlap in nucleotides of replicate 1 and replicate 2.  
 "overlap" the interaction pair is chosen which has the highest relative overlap, i.e., overlap in nucleotides of replicate 1 and replicate 2.  
 "midpoint" the interaction pair is chosen which has the smallest distance between their anchor midpoints, i.e., distance between the midpoint of the two anchors.

max\_gap integer; maximum gap in nucleotides allowed between two anchors for them to be considered as overlapping (defaults to -1, i.e., overlapping anchors)

## Value

data frame with the following columns:

column 1: rep1\_idx index of interaction in replicate 1 (i.e., row index in rep1\_df)  
 column 2: rep2\_idx index of interaction in replicate 2 (i.e., row index in rep2\_df)  
 column 3: arv ambiguity resolution value used turn m:n mapping into 1:1 mapping. Interaction pairs with lower arv are prioritized.

## Examples

```
rep1_df <- idr2d::chiapet$rep1_df
rep1_df$fdr <- preprocess(rep1_df$fdr, "log_additive_inverse")

rep2_df <- idr2d::chiapet$rep2_df
rep2_df$fdr <- preprocess(rep2_df$fdr, "log_additive_inverse")

# shuffle to break preexisting order
rep1_df <- rep1_df[sample.int(nrow(rep1_df)), ]
rep2_df <- rep2_df[sample.int(nrow(rep2_df)), ]

# sort by value column
rep1_df <- dplyr::arrange(rep1_df, rep1_df$fdr)
rep2_df <- dplyr::arrange(rep2_df, rep2_df$fdr)

pairs_df <- establish_overlap2d(rep1_df, rep2_df)
```

estimate\_idr

*Estimates IDR for Genomic Peaks or Genomic Interactions***Description**

Estimates IDR for Genomic Peaks or Genomic Interactions

**Usage**

```
estimate_idr(
  rep1_df,
  rep2_df,
  analysis_type = "IDR2D",
  value_transformation = c("identity", "additive_inverse", "multiplicative_inverse",
    "log", "log_additive_inverse"),
  ambiguity_resolution_method = c("overlap", "midpoint", "value"),
  remove_nonstandard_chromosomes = TRUE,
  max_factor = 1.5,
  jitter_factor = 1e-04,
  max_gap = -1L,
  mu = 0.1,
  sigma = 1,
  rho = 0.2,
  p = 0.5,
  eps = 0.001,
  max_iteration = 30,
  local_idr = TRUE
)
```

**Arguments**

rep1\_df data frame of observations (i.e., genomic peaks or genomic interactions) of replicate 1. If analysis\_type is IDR1D, the columns of rep1\_df are described in [establish\\_bijection1d](#), otherwise in [establish\\_bijection2d](#)

rep2\_df data frame of observations (i.e., genomic peaks or genomic interactions) of replicate 2. Same columns as rep1\_df.

analysis\_type "IDR2D" for genomic interaction data sets, "IDR1D" for genomic peak data sets  
value\_transformation

the values in x have to be transformed in a way such that when ordered in descending order, more significant interactions end up on top of the list. If the values in x are p-values, "log\_additive\_inverse" is recommended. The following transformations are supported:

"identity"	no transformation is performed on x
"additive_inverse"	$x. = -x$
"multiplicative_inverse"	$x. = 1 / x$
"log"	$x. = \log(x)$ . Note: zeros are replaced by <code>.Machine\$double.xmin</code>
"log_additive_inverse"	$x. = -\log(x)$ , recommended if x are p-values. Note: zeros are replaced by <code>.Machine\$double.xmin</code>

either "ascending" (more significant interactions have lower value in value column) or "descending" (more significant interactions have higher value in value column)

ambiguity_resolution_method	defines how ambiguous assignments (when one interaction or peak in replicate 1 overlaps with multiple interactions or peaks in replicate 2 or vice versa) are resolved. For available methods, see <a href="#">establish_overlap1d</a> or <a href="#">establish_overlap2d</a> , respectively.
remove_nonstandard_chromosomes	removes peaks and interactions containing genomic locations on non-standard chromosomes using <a href="#">keepStandardChromosomes</a> (default is TRUE)
max_factor	numeric; controls the replacement values for Inf and -Inf. Inf are replaced by $\max(x) * \text{max\_factor}$ and -Inf are replaced by $\min(x) / \text{max\_factor}$ .
jitter_factor	numeric; controls the magnitude of the noise that is added to x. This is done to break ties in x. Set jitter_factor = NULL for no jitter.
max_gap	integer; maximum gap in nucleotides allowed between two anchors for them to be considered as overlapping (defaults to -1, i.e., overlapping anchors)
mu	a starting value for the mean of the reproducible component.
sigma	a starting value for the standard deviation of the reproducible component.
rho	a starting value for the correlation coefficient of the reproducible component.
p	a starting value for the proportion of reproducible component.
eps	Stopping criterion. Iterations stop when the increment of log-likelihood is $< \text{eps} * \log\text{-likelihood}$ , Default=0.001.
max_iteration	integer; maximum number of iterations for IDR estimation (defaults to 30)
local_idr	see <a href="#">est.IDR</a>

**Value**

See [estimate\\_idr1d](#) or [estimate\\_idr2d](#), respectively.

**References**

Q. Li, J. B. Brown, H. Huang and P. J. Bickel. (2011) Measuring reproducibility of high-throughput experiments. *Annals of Applied Statistics*, Vol. 5, No. 3, 1752-1779.

**Examples**

```
idr_results <- estimate_idr(idr2d:::chiapet$rep1_df,
                           idr2d:::chiapet$rep2_df,
                           analysis_type = "IDR2D",
                           value_transformation = "log_additive_inverse")
summary(idr_results)
```

---

estimate\_idr1d

*Estimates IDR for Genomic Peak Data*

---

**Description**

This method estimates Irreproducible Discovery Rates (IDR) for peaks in replicated ChIP-seq experiments.

**Usage**

```
estimate_idr1d(
  rep1_df,
  rep2_df,
  value_transformation = c("identity", "additive_inverse", "multiplicative_inverse",
    "log", "log_additive_inverse"),
  ambiguity_resolution_method = c("overlap", "midpoint", "value"),
  remove_nonstandard_chromosomes = TRUE,
  max_factor = 1.5,
  jitter_factor = 1e-04,
  max_gap = -1L,
  mu = 0.1,
  sigma = 1,
  rho = 0.2,
  p = 0.5,
  eps = 0.001,
  max_iteration = 30,
  local_idr = TRUE
)
```

**Arguments**

`rep1_df` data frame of observations (i.e., genomic peaks) of replicate 1, with at least the following columns (position of columns matter, column names are irrelevant):

column 1:	<code>chr</code>	character; genomic location of peak - chromosome (e.g., "chr3")
column 2:	<code>start</code>	integer; genomic location of peak - start coordinate
column 3:	<code>end</code>	integer; genomic location of peak - end coordinate
column 4:	<code>value</code>	numeric; p-value, FDR, or heuristic used to rank the interactions

`rep2_df` data frame of observations (i.e., genomic peaks) of replicate 2, with the following columns (position of columns matter, column names are irrelevant):

column 1:	<code>chr</code>	character; genomic location of peak - chromosome (e.g., "chr3")
column 2:	<code>start</code>	integer; genomic location of peak - start coordinate
column 3:	<code>end</code>	integer; genomic location of peak - end coordinate
column 4:	<code>value</code>	numeric; p-value, FDR, or heuristic used to rank the interactions

`value_transformation`

the values in `x` have to be transformed in a way such that when ordered in descending order, more significant interactions end up on top of the list. If the values in `x` are p-values, "log\_additive\_inverse" is recommended. The following transformations are supported:

"identity"	no transformation is performed on <code>x</code>
"additive_inverse"	$x. = -x$
"multiplicative_inverse"	$x. = 1 / x$
"log"	$x. = \log(x)$ . Note: zeros are replaced by <code>.Machine\$double.xmin</code>
"log_additive_inverse"	$x. = -\log(x)$ , recommended if <code>x</code> are p-values. Note: zeros are replaced by <code>.Machine\$double.xmin</code>

either "ascending" (more significant interactions have lower value in value column) or "descending" (more significant interactions have higher value in value column)

ambiguity_resolution_method	defines how ambiguous assignments (when one interaction in replicate 1 overlaps with multiple interactions in replicate 2 or vice versa) are resolved. Available methods:
"value"	interactions are prioritized by ascending or descending value column (see <code>sorting_direction</code> ), e.g., if <code>ascending</code>
"overlap"	the interaction pair is chosen which has the highest relative overlap, i.e., overlap in nucleotides of replicate 1 and 2
"midpoint"	the interaction pair is chosen which has the smallest distance between their anchor midpoints, i.e., distance between anchors
remove_nonstandard_chromosomes	removes peaks containing genomic locations on non-standard chromosomes using <code>keepStandardChromosomes</code> (default is TRUE)
max_factor	numeric; controls the replacement values for Inf and -Inf. Inf are replaced by $\max(x) * \text{max\_factor}$ and -Inf are replaced by $\min(x) / \text{max\_factor}$ .
jitter_factor	numeric; controls the magnitude of the noise that is added to x. This is done to break ties in x. Set <code>jitter_factor = NULL</code> for no jitter.
max_gap	integer; maximum gap in nucleotides allowed between two anchors for them to be considered as overlapping (defaults to -1, i.e., overlapping anchors)
mu	a starting value for the mean of the reproducible component.
sigma	a starting value for the standard deviation of the reproducible component.
rho	a starting value for the correlation coefficient of the reproducible component.
p	a starting value for the proportion of reproducible component.
eps	Stopping criterion. Iterations stop when the increment of log-likelihood is $< \text{eps} * \log\text{-likelihood}$ , Default=0.001.
max_iteration	integer; maximum number of iterations for IDR estimation (defaults to 30)
local_idr	see <code>est.IDR</code>

## Value

List with three components, (`rep1_df`, `rep2_df`, and `analysis_type`) containing the interactions from input data frames `rep1_df` and `rep2_df` with the following additional columns:

column 1:	<code>chr</code>	character; genomic location of peak - chromosome (e.g., "chr3")
column 2:	<code>start</code>	integer; genomic location of peak - start coordinate
column 3:	<code>end</code>	integer; genomic location of peak - end coordinate
column 4:	<code>value</code>	numeric; p-value, FDR, or heuristic used to rank the peaks
column 5:	<code>rep_value</code>	numeric; value of corresponding replicate peak. If no corresponding peak was found, <code>rep_value = NA</code>
column 6:	<code>rank</code>	integer; rank of the peak, established by value column, ascending order
column 7:	<code>rep_rank</code>	integer; rank of corresponding replicate peak. If no corresponding peak was found, <code>rep_rank = NA</code>
column 8:	<code>idx</code>	integer; peak index, primary key
column 9:	<code>rep_idx</code>	integer; specifies the index of the corresponding peak in the other replicate (foreign key). If no corresponding peak was found, <code>rep_idx = NA</code>
column 10:	<code>idr</code>	IDR of the peak and the corresponding peak in the other replicate. If no corresponding peak was found, <code>idr = NA</code>

## References

Q. Li, J. B. Brown, H. Huang and P. J. Bickel. (2011) Measuring reproducibility of high-throughput experiments. *Annals of Applied Statistics*, Vol. 5, No. 3, 1752-1779.

**Examples**

```
idr_results <- estimate_idr1d(idr2d::chipseq$rep1_df,
                             idr2d::chipseq$rep2_df,
                             value_transformation = "log")
summary(idr_results)
```

estimate\_idr2d

*Estimates IDR for Genomic Interaction Data***Description**

This method estimates Irreproducible Discovery Rates (IDR) between two replicates of experiments identifying genomic interactions, such as Hi-C, ChIA-PET, and HiChIP.

**Usage**

```
estimate_idr2d(
  rep1_df,
  rep2_df,
  value_transformation = c("identity", "additive_inverse", "multiplicative_inverse",
    "log", "log_additive_inverse"),
  ambiguity_resolution_method = c("overlap", "midpoint", "value"),
  remove_nonstandard_chromosomes = TRUE,
  max_factor = 1.5,
  jitter_factor = 1e-04,
  max_gap = -1L,
  mu = 0.1,
  sigma = 1,
  rho = 0.2,
  p = 0.5,
  eps = 0.001,
  max_iteration = 30,
  local_idr = TRUE
)
```

**Arguments**

rep1\_df            data frame of observations (i.e., genomic interactions) of replicate 1, with at least the following columns (position of columns matter, column names are irrelevant):

column 1:	chr_a	character; genomic location of anchor A - chromosome (e.g., "chr3")
column 2:	start_a	integer; genomic location of anchor A - start coordinate
column 3:	end_a	integer; genomic location of anchor A - end coordinate
column 4:	chr_b	character; genomic location of anchor B - chromosome (e.g., "chr3")
column 5:	start_b	integer; genomic location of anchor B - start coordinate
column 6:	end_b	integer; genomic location of anchor B - end coordinate
column 7:	value	numeric; p-value, FDR, or heuristic used to rank the interactions

rep2\_df data frame of observations (i.e., genomic interactions) of replicate 2, with the following columns (position of columns matter, column names are irrelevant):

column 1: chr\_a character; genomic location of anchor A - chromosome (e.g., "chr3")  
 column 2: start\_a integer; genomic location of anchor A - start coordinate  
 column 3: end\_a integer; genomic location of anchor A - end coordinate  
 column 4: chr\_b character; genomic location of anchor B - chromosome (e.g., "chr3")  
 column 5: start\_b integer; genomic location of anchor B - start coordinate  
 column 6: end\_b integer; genomic location of anchor B - end coordinate  
 column 7: value numeric; p-value, FDR, or heuristic used to rank the interactions

value\_transformation

the values in  $x$  have to be transformed in a way such that when ordered in descending order, more significant interactions end up on top of the list. If the values in  $x$  are p-values, "log\_additive\_inverse" is recommended. The following transformations are supported:

"identity" no transformation is performed on  $x$   
 "additive\_inverse"  $x. = -x$   
 "multiplicative\_inverse"  $x. = 1 / x$   
 "log"  $x. = \log(x)$ . Note: zeros are replaced by `.Machine$double.xmin`  
 "log\_additive\_inverse"  $x. = -\log(x)$ , recommended if  $x$  are p-values. Note: zeros are replaced by `.Machine$double.xmin`

either "ascending" (more significant interactions have lower value in value column) or "descending" (more significant interactions have higher value in value column)

ambiguity\_resolution\_method

defines how ambiguous assignments (when one interaction in replicate 1 overlaps with multiple interactions in replicate 2 or vice versa) are resolved. Available methods:

"value" interactions are prioritized by ascending or descending value column (see `sorting_direction`), e.g., if `value` is "ascending",  
 "overlap" the interaction pair is chosen which has the highest relative overlap, i.e., overlap in nucleotides of replicate 1 and replicate 2  
 "midpoint" the interaction pair is chosen which has the smallest distance between their anchor midpoints, i.e., distance between anchor midpoints

remove\_nonstandard\_chromosomes

removes interactions containing genomic locations on non-standard chromosomes using `keepStandardChromosomes` (default is TRUE)

max\_factor numeric; controls the replacement values for  $\text{Inf}$  and  $-\text{Inf}$ .  $\text{Inf}$  are replaced by  $\max(x) * \text{max\_factor}$  and  $-\text{Inf}$  are replaced by  $\min(x) / \text{max\_factor}$ .

jitter\_factor numeric; controls the magnitude of the noise that is added to  $x$ . This is done to break ties in  $x$ . Set `jitter_factor = NULL` for no jitter.

max\_gap integer; maximum gap in nucleotides allowed between two anchors for them to be considered as overlapping (defaults to -1, i.e., overlapping anchors)

mu a starting value for the mean of the reproducible component.

sigma a starting value for the standard deviation of the reproducible component.

rho a starting value for the correlation coefficient of the reproducible component.

p a starting value for the proportion of reproducible component.

eps Stopping criterion. Iterations stop when the increment of log-likelihood is  $< \text{eps} * \log\text{-likelihood}$ , Default=0.001.

max\_iteration integer; maximum number of iterations for IDR estimation (defaults to 30)

local\_idr see [est.IDR](#)

**Value**

List with three components, (rep1\_df, rep2\_df, and analysis\_type) containing the interactions from input data frames rep1\_df and rep2\_df with the following additional columns:

```
column 1: chr_a
column 2: start_a
column 3: end_a
column 4: chr_b
column 5: start_b
column 6: end_b
column 7: value
column 8: "rep_value"
column 9: "rank"
column 10: "rep_rank"
column 11: "idx"
column 12: "rep_idx"
idr      IDR of the interaction and the corresponding interaction in the other replicate. If no corresponding interaction
```

**References**

Q. Li, J. B. Brown, H. Huang and P. J. Bickel. (2011) Measuring reproducibility of high-throughput experiments. *Annals of Applied Statistics*, Vol. 5, No. 3, 1752-1779.

**Examples**

```
idr_results <- estimate_idr2d(idr2d::chiapet$rep1_df,
                             idr2d::chiapet$rep2_df,
                             value_transformation = "log_additive_inverse")
summary(idr_results)
```

---

estimate_idr2d_hic	<i>Estimates IDR for Genomic Interactions measured by Hi-C experiments</i>
--------------------	--

---

**Description**

This method estimates Irreproducible Discovery Rates (IDR) of genomic interactions between two replicates of Hi-C experiments.

Before calling this method, call Juicer .hic contact matrix c

The contact matrix is subdivided into blocks, where the block size is determined by resolution. The reads per block are used to rank blocks and replicate blocks are easily matched by genomic location.

**Usage**

```
estimate_idr2d_hic(
  rep1_df,
  rep2_df,
  combined_min_value = 30,
  combined_max_value = Inf,
```

```

min_value = -Inf,
max_value = Inf,
max_factor = 1.5,
jitter_factor = 1e-04,
mu = 0.1,
sigma = 1,
rho = 0.2,
p = 0.5,
eps = 0.001,
max_iteration = 30,
local_idr = TRUE
)

```

### Arguments

rep1_df	data frame of either parsed .hic file from Juicer (output of <a href="#">parse_juicer_matrix</a> ) or parsed .matrix and .bed files from HiC-Pro (output of <a href="#">parse_hic_pro_matrix</a> ) for replicate 1
rep2_df	data frame of either parsed .hic file from Juicer (output of <a href="#">parse_juicer_matrix</a> ) or parsed .matrix and .bed files from HiC-Pro (output of <a href="#">parse_hic_pro_matrix</a> ) for replicate 2
combined_min_value	exclude blocks with a combined (replicate 1 + replicate 2) read count or normalized read count of less than combined_min_value (default is 20 reads, set combined_min_value = -Inf to disable)
combined_max_value	exclude blocks with a combined (replicate 1 + replicate 2) read count or normalized read count of more than combined_max_value (disabled by default, set combined_max_value = Inf to disable)
min_value	exclude blocks with a read count or normalized read count of less than min_value in one replicate (disabled by default, set min_value = -Inf to disable)
max_value	exclude blocks with a read count or normalized read count of more than max_value in one replicate (disabled by default, set max_value = Inf to disable)
max_factor	numeric; controls the replacement values for Inf and -Inf. Inf are replaced by $\max(x) * \text{max\_factor}$ and -Inf are replaced by $\min(x) / \text{max\_factor}$ .
jitter_factor	numeric; controls the magnitude of the noise that is added to x. This is done to break ties in x. Set jitter_factor = NULL for no jitter.
mu	a starting value for the mean of the reproducible component.
sigma	a starting value for the standard deviation of the reproducible component.
rho	a starting value for the correlation coefficient of the reproducible component.
p	a starting value for the proportion of reproducible component.
eps	Stopping criterion. Iterations stop when the increment of log-likelihood is $< \text{eps} * \text{log-likelihood}$ , Default=0.001.
max_iteration	integer; maximum number of iterations for IDR estimation (defaults to 30)
local_idr	see <a href="#">est.IDR</a>

**Value**

Data frame with the following columns:

column 1:	interaction	character; genomic location of interaction block (e.g., "chr1:204940000-204940000")
column 2:	value	numeric; p-value, FDR, or heuristic used to rank the interactions
column 3:	"rep_value"	numeric; value of corresponding replicate interaction
column 4:	"rank"	integer; rank of the interaction, established by value column, ascending order
column 5:	"rep_rank"	integer; rank of corresponding replicate interaction
column 6:	"idr"	integer; IDR of the block and the corresponding block in the other replicate

**References**

Q. Li, J. B. Brown, H. Huang and P. J. Bickel. (2011) Measuring reproducibility of high-throughput experiments. *Annals of Applied Statistics*, Vol. 5, No. 3, 1752-1779.

**Examples**

```
idr_results_df <- estimate_idr2d_hic(idr2d:::hic$rep1_df,
                                   idr2d:::hic$rep2_df)
summary(idr_results_df)
```

---

hic

*Example Hi-C data set*

---

**Description**

This object contains data from a Hi-C contact map of human chromosome 1 and a resolution of  $2.5 * 10^6$ , extracted from GEO series GSE71831.

**Usage**

```
hic
```

**Format**

A list with two components, the data frames rep1\_df and rep2\_df, which have the following four columns:

column 1:	chr	character; genomic location of block - chromosome (e.g., "chr3")
column 2:	region1	integer; genomic location of block - coordinate A
column 3:	region2	integer; genomic location of block - coordinate B
column 4:	value	numeric; heuristic used to rank blocks, in this case: number of reads

---

parse\_hic\_pro\_matrix *Parse .matrix and .bed files from HiC-Pro for IDR2D analysis*

---

### Description

This function is used to convert the contact matrix from a HiC-Pro pipeline analysis run into an IDR2D compatible format. It takes one .matrix and one .bed file per replicate from HiC-Pro and returns the contact matrix for a specific chromosome for IDR2D analysis (see [estimate\\_idr2d\\_hic](#))

### Usage

```
parse_hic_pro_matrix(matrix_file, bed_file, chromosome = "chr1")
```

### Arguments

matrix_file	path to .matrix file from HiC-Pro analysis run
bed_file	path to .bed file from HiC-Pro analysis run
chromosome	chromosome name to be analyzed, defaults to UCSC chromosome 1 ("chr1")

### Value

Data frame with the following columns:

column 1:	chr	character; chromosome of block (e.g., "chr3")
column 2:	region1	integer; genomic location of side A of block in Hi-C contact matrix
column 3:	region2	integer; genomic location of side B of block in Hi-C contact matrix
column 4:	value	numeric; (normalized) read count in block

### References

Servant, N., Varoquaux, N., Lajoie, B.R. et al. HiC-Pro: an optimized and flexible pipeline for Hi-C data processing. *Genome Biol* 16, 259 (2015) doi:10.1186/s13059-015-0831-x

---

parse\_juicer\_matrix *Parse .hic files from Juicer for IDR2D analysis*

---

### Description

parse\_juicer\_matrix uses the Python package `hic-straw` internally to read .hic contact matrix files (see [hic-straw on PyPI](#) or the [Aiden lab GitHub repository](#) for more information).

The contact matrix is subdivided into blocks, where the block size is determined by resolution. The reads per block are used to rank blocks and replicate blocks are easily matched by genomic location.

**Usage**

```

parse_juicer_matrix(
  hic_file,
  resolution = 1e+06,
  normalization = c("NONE", "VC", "VC_SQRT", "KR"),
  chromosome = "chr1",
  use_python = NULL,
  use_virtualenv = NULL,
  use_condaenv = NULL
)

```

**Arguments**

<code>hic_file</code>	path to .hic file (either local file path or URL).
<code>resolution</code>	block resolution of Hi-C contact matrix in base pairs, defaults to 1,000,000 bp (usually one of the following: 2500000, 1000000, 500000, 250000, 100000, 50000, 25000, 10000, 5000)
<code>normalization</code>	normalization step performed by Python package <code>hic-straw</code> , one of the following: "NONE", "VC", "VC_SQRT", "KR".
<code>chromosome</code>	chromosome name to be analyzed, defaults to UCSC chromosome 1 ("chr1")
<code>use_python</code>	if Python is not on PATH, specify path to Python binary here (see <a href="#">use_python</a> )
<code>use_virtualenv</code>	if Python package <code>hic-straw</code> is not in base virtualenv environment, specify environment here (see <a href="#">use_virtualenv</a> )
<code>use_condaenv</code>	if Python package <code>hic-straw</code> is not in base conda environment, specify environment here (see <a href="#">use_condaenv</a> )

**Value**

Data frame with the following columns:

column 1:	<code>chr</code>	character; chromosome of block (e.g., "chr3")
column 2:	<code>region1</code>	integer; genomic location of side A of block in Hi-C contact matrix
column 3:	<code>region2</code>	integer; genomic location of side B of block in Hi-C contact matrix
column 4:	<code>value</code>	numeric; (normalized) read count in block

**References**

Neva C. Durand, James T. Robinson, Muhammad S. Shamim, Ido Machol, Jill P. Mesirov, Eric S. Lander, and Erez Lieberman Aiden. "Juicebox provides a visualization system for Hi-C contact maps with unlimited zoom." *Cell Systems* 3(1), 2016.

**Description**

This method removes invalid values, establishes the correct ranking, and breaks ties prior to IDR analysis.

Inf and -Inf are replaced by  $\max(x) * \text{max\_factor}$  and  $\min(x) / \text{max\_factor}$ , respectively.

NA values in  $x$  are replaced by  $\text{mean}(x)$ .

All values in  $x$  are transformed using the transformation specified in `value_transformation`.

Lastly, a small amount of noise is added to  $x$  to break ties. The magnitude of the noise is controlled by `jitter_factor`.

**Usage**

```
preprocess(
  x,
  value_transformation = c("identity", "additive_inverse", "multiplicative_inverse",
    "log", "log_additive_inverse"),
  max_factor = 1.5,
  jitter_factor = 1e-04
)
```

**Arguments**

<code>x</code>	numeric vector of values										
<code>value_transformation</code>	the values in $x$ have to be transformed in a way such that when ordered in descending order, more significant interactions end up on top of the list. If the values in $x$ are p-values, "log_additive_inverse" is recommended. The following transformations are supported:										
	<table> <tr> <td>"identity"</td> <td>no transformation is performed on <math>x</math></td> </tr> <tr> <td>"additive_inverse"</td> <td><math>x. = -x</math></td> </tr> <tr> <td>"multiplicative_inverse"</td> <td><math>x. = 1 / x</math></td> </tr> <tr> <td>"log"</td> <td><math>x. = \log(x)</math>. Note: zeros are replaced by <code>.Machine\$double.xmin</code></td> </tr> <tr> <td>"log_additive_inverse"</td> <td><math>x. = -\log(x)</math>, recommended if <math>x</math> are p-values. Note: zeros are replaced by <code>.Machine\$double.xmin</code></td> </tr> </table>	"identity"	no transformation is performed on $x$	"additive_inverse"	$x. = -x$	"multiplicative_inverse"	$x. = 1 / x$	"log"	$x. = \log(x)$ . Note: zeros are replaced by <code>.Machine\$double.xmin</code>	"log_additive_inverse"	$x. = -\log(x)$ , recommended if $x$ are p-values. Note: zeros are replaced by <code>.Machine\$double.xmin</code>
"identity"	no transformation is performed on $x$										
"additive_inverse"	$x. = -x$										
"multiplicative_inverse"	$x. = 1 / x$										
"log"	$x. = \log(x)$ . Note: zeros are replaced by <code>.Machine\$double.xmin</code>										
"log_additive_inverse"	$x. = -\log(x)$ , recommended if $x$ are p-values. Note: zeros are replaced by <code>.Machine\$double.xmin</code>										
	either "ascending" (more significant interactions have lower value in value column) or "descending" (more significant interactions have higher value in value column)										
<code>max_factor</code>	numeric; controls the replacement values for Inf and -Inf. Inf are replaced by $\max(x) * \text{max\_factor}$ and -Inf are replaced by $\min(x) / \text{max\_factor}$ .										
<code>jitter_factor</code>	numeric; controls the magnitude of the noise that is added to $x$ . This is done to break ties in $x$ . Set <code>jitter_factor = NULL</code> for no jitter.										

**Value**

numeric vector; transformed and stripped values of  $x$ , ready for IDR analysis

**Examples**

```
rep1_df <- idr2d::chiapet$rep1_df
rep1_df$fdr <- preprocess(rep1_df$fdr, "log_additive_inverse")
```

---

`remove_nonstandard_chromosomes1d`*Removes Peaks on Non-standard Chromosomes*

---

**Description**

Removes Peaks on Non-standard Chromosomes

**Usage**`remove_nonstandard_chromosomes1d(x)`**Arguments**

`x` data frame of genomic peaks, with the following columns (position of columns matter, column names are irrelevant):

column 1:	<code>chr</code>	character; genomic location of peak - chromosome (e.g., "chr3")
column 2:	<code>start</code>	integer; genomic location of peak - start coordinate
column 3:	<code>end</code>	integer; genomic location of peak - end coordinate
column 4:	<code>value</code>	numeric; p-value, FDR, or heuristic used to rank the peaks

**Value**`x` without non-standard chromosomes.**Examples**

```
rep1_df <- remove_nonstandard_chromosomes1d(idr2d:::chipseq$rep1_df)
```

---

`remove_nonstandard_chromosomes2d`*Removes Interactions on Non-standard Chromosomes*

---

**Description**

Removes Interactions on Non-standard Chromosomes

**Usage**`remove_nonstandard_chromosomes2d(x)`

**Arguments**

`x` data frame of genomic interactions, with the following columns (position of columns matter, column names are irrelevant):

column 1:	<code>chr_a</code>	character; genomic location of anchor A - chromosome (e.g., "chr3")
column 2:	<code>start_a</code>	integer; genomic location of anchor A - start coordinate
column 3:	<code>end_a</code>	integer; genomic location of anchor A - end coordinate
column 4:	<code>chr_b</code>	character; genomic location of anchor B - chromosome (e.g., "chr3")
column 5:	<code>start_b</code>	integer; genomic location of anchor B - start coordinate
column 6:	<code>end_b</code>	integer; genomic location of anchor B - end coordinate
column 7:	<code>value</code>	numeric; p-value, FDR, or heuristic used to rank the interactions

**Value**

`x` without non-standard chromosomes.

**Examples**

```
rep1_df <- remove_nonstandard_chromosomes2d(idr2d:::chiapet$rep1_df)
```

# Index

## \* datasets

- chiapet, [7](#)
- chipseq, [8](#)
- hic, [29](#)

- calculate\_midpoint\_distance1d, [2](#)
- calculate\_midpoint\_distance2d, [3](#)
- calculate\_relative\_overlap1d, [5](#)
- calculate\_relative\_overlap2d, [6](#)
- chiapet, [7](#)
- chipseq, [8](#)

- determine\_anchor\_overlap, [8](#)
- draw\_hic\_contact\_map, [9](#)
- draw\_idr\_distribution\_histogram, [10](#)
- draw\_rank\_idr\_scatterplot, [11](#)
- draw\_value\_idr\_scatterplot, [12](#)

- est.IDR, [22](#), [24](#), [26](#), [28](#)
- establish\_bijection, [13](#)
- establish\_bijection1d, [14](#), [14](#), [21](#)
- establish\_bijection2d, [14](#), [16](#), [21](#)
- establish\_overlap1d, [14](#), [17](#), [22](#)
- establish\_overlap2d, [14](#), [19](#), [22](#)
- estimate\_idr, [10–13](#), [21](#)
- estimate\_idr1d, [22](#), [22](#)
- estimate\_idr2d, [22](#), [25](#)
- estimate\_idr2d\_hic, [9](#), [27](#), [30](#)

- hic, [29](#)

- keepStandardChromosomes, [22](#), [24](#), [26](#)

- parse\_hic\_pro\_matrix, [28](#), [30](#)
- parse\_juicer\_matrix, [28](#), [30](#)
- preprocess, [31](#)

- remove\_nonstandard\_chromosomes1d, [33](#)
- remove\_nonstandard\_chromosomes2d, [33](#)

- use\_condaenv, [31](#)
- use\_python, [31](#)
- use\_virtualenv, [31](#)