

# Package ‘redist’

February 12, 2026

**Version** 4.3.2

**Date** 2026-02-10

**Title** Simulation Methods for Legislative Redistricting

**Description** Enables researchers to sample redistricting plans from a pre-specified target distribution using Sequential Monte Carlo and Markov Chain Monte Carlo algorithms. The package allows for the implementation of various constraints in the redistricting process such as geographic compactness and population parity requirements. Tools for analysis such as computation of various summary statistics and plotting functionality are also included. The package implements the SMC algorithm of McCartan and Imai (2023) <[doi:10.1214/23-AOAS1763](https://doi.org/10.1214/23-AOAS1763)>, the enumeration algorithm of Fifield, Imai, Kawahara, and Kenny (2020) <[doi:10.1080/2330443X.2020.1791773](https://doi.org/10.1080/2330443X.2020.1791773)>, the Flip MCMC algorithm of Fifield, Higgins, Imai and Tarr (2020) <[doi:10.1080/10618600.2020.1739532](https://doi.org/10.1080/10618600.2020.1739532)>, the Merge-split/Recombination algorithms of Carter et al. (2019) <[doi:10.48550/arXiv.1911.01503](https://doi.org/10.48550/arXiv.1911.01503)> and DeFord et al. (2021) <[doi:10.1162/99608f92.eb30390f](https://doi.org/10.1162/99608f92.eb30390f)>, and the Short-burst optimization algorithm of Cannon et al. (2020) <[doi:10.48550/arXiv.2011.02288](https://doi.org/10.48550/arXiv.2011.02288)>.

**Depends** R (>= 4.1.0), redistmetrics (>= 1.0.2)

**Imports** Rcpp (>= 0.11.0), rlang, cli (>= 3.1.0), vctrs, tidyselect, stringr, dplyr (>= 1.0.0), sf, doParallel, foreach, doRNG, servr, sys, ggplot2, patchwork

**Suggests** coda, matrixStats, loo, Rmpi, withr, knitr, rmarkdown, rmapshaper, ggpattern, scales, units, RSpectra, testthat (>= 3.0.0), spelling

**LinkingTo** Rcpp, RcppArmadillo, RcppThread, cli, redistmetrics

**License** GPL (>= 2)

**SystemRequirements** C++17, python

**NeedsCompilation** yes

**BugReports** <https://github.com/alarm-redist/redist/issues>

**URL** <https://alarm-redist.org/redist/>

**RoxygenNote** 7.3.3

**VignetteBuilder** knitr

**Encoding** UTF-8

**Config/testthat/edition** 3

**Language** en-US

**LazyData** true

**Author** Christopher T. Kenny [aut, cre] (ORCID:

<<https://orcid.org/0000-0002-9386-6860>>),

Cory McCartan [aut] (ORCID: <<https://orcid.org/0000-0002-6251-669X>>),

Ben Fifield [aut] (ORCID: <<https://orcid.org/0000-0002-2247-0201>>),

Kosuke Imai [aut] (ORCID: <<https://orcid.org/0000-0002-2748-1022>>),

Jun Kawahara [ctb],

Alexander Tarr [ctb],

Michael Higgins [ctb]

**Maintainer** Christopher T. Kenny <ctkenny@proton.me>

**Repository** CRAN

**Date/Publication** 2026-02-12 16:30:02 UTC

## Contents

add_reference . . . . .	5
avg_by_prec . . . . .	5
classify_plans . . . . .	6
compare_plans . . . . .	6
competitiveness . . . . .	8
constraints . . . . .	9
county_splits . . . . .	13
distr_compactness . . . . .	13
EPSG . . . . .	17
fl25 . . . . .	17
fl250 . . . . .	18
fl25_adj . . . . .	19
fl25_enum . . . . .	20
fl70 . . . . .	20
freeze . . . . .	21
get_adj . . . . .	22
get_existing . . . . .	23
get_mh_acceptance_rate . . . . .	23
get_plans_matrix . . . . .	24
get_plans_weights . . . . .	24
get_pop_tol . . . . .	25
get_sampling_info . . . . .	25
get_target . . . . .	26
group_frac . . . . .	26
iowa . . . . .	27
is_contiguous . . . . .	28

is_county_split . . . . .	28
last_plan . . . . .	29
make_cores . . . . .	29
match_numbers . . . . .	30
merge_by . . . . .	31
min_move_parity . . . . .	32
muni_splits . . . . .	33
number_by . . . . .	33
partisan_metrics . . . . .	34
pl . . . . .	36
plans_diversity . . . . .	37
plan_distances . . . . .	38
plot.redist_classified . . . . .	39
plot.redist_constr . . . . .	40
plot.redist_map . . . . .	41
plot.redist_plans . . . . .	42
prec_assignment . . . . .	42
prec_cooccurrence . . . . .	43
print.redist_classified . . . . .	43
print.redist_constr . . . . .	44
print.redist_map . . . . .	44
print.redist_plans . . . . .	45
proj . . . . .	45
pullback . . . . .	47
rbind.redist_plans . . . . .	47
redist.adjacency . . . . .	48
redist.calc.frontier.size . . . . .	48
redist.coarsen.adjacency . . . . .	49
redist.combine.mpi . . . . .	49
redist.constraint.helper . . . . .	51
redist.county.id . . . . .	52
redist.county.relabel . . . . .	53
redist.crs . . . . .	53
redist.diagplot . . . . .	55
redist.dist.pop.overlap . . . . .	57
redist.district_splits . . . . .	58
redist.enumpart . . . . .	59
redist.find.target . . . . .	60
redist.findparams . . . . .	60
redist.init.enumpart . . . . .	63
redist.ipw . . . . .	63
redist.mcmc.mpi . . . . .	66
redist.multisplits . . . . .	69
redist.parity . . . . .	70
redist.plot.adj . . . . .	70
redist.plot.contr_pfd . . . . .	71
redist.plot.cores . . . . .	72
redist.plot.distr_qtys . . . . .	73

redist.plot.hist . . . . .	75
redist.plot.majmin . . . . .	76
redist.plot.map . . . . .	76
redist.plot.penalty . . . . .	77
redist.plot.plans . . . . .	78
redist.plot.scatter . . . . .	79
redist.plot.trace . . . . .	80
redist.plot.varinfo . . . . .	81
redist.plot.wted.adj . . . . .	81
redist.prec.pop.overlap . . . . .	82
redist.prep.enumpart . . . . .	83
redist.random.subgraph . . . . .	84
redist.read.enumpart . . . . .	85
redist.reduce.adjacency . . . . .	86
redist.reorder . . . . .	86
redist.rsg . . . . .	87
redist.run.enumpart . . . . .	88
redist.sink.plan . . . . .	89
redist.smc_is_ci . . . . .	90
redist.subset . . . . .	91
redist.uncoarsen . . . . .	92
redist.wted.adj . . . . .	92
redist_ci . . . . .	93
redist_constr . . . . .	94
redist_flip . . . . .	95
redist_flip_anneal . . . . .	99
redist_map . . . . .	101
redist_mergesplit . . . . .	102
redist_mergesplit_parallel . . . . .	105
redist_plans . . . . .	107
redist_quantile_trunc . . . . .	109
redist_shortburst . . . . .	109
redist_smc . . . . .	111
scorer-arith . . . . .	114
scorer-combine . . . . .	115
scorer_group_pct . . . . .	116
segregation_index . . . . .	117
subset_sampled . . . . .	119
summary.redist_plans . . . . .	119
tally_var . . . . .	121

---

add_reference	<i>Add a reference plan to a set of plans</i>
---------------	---

---

**Description**

This function facilitates comparing an existing (i.e., non-simulated) redistricting plan to a set of simulated plans.

**Usage**

```
add_reference(plans, ref_plan, name = NULL)
```

**Arguments**

plans	a <code>redist_plans</code> object
ref_plan	an integer vector containing the reference plan. It will be renumbered to <code>1:nplans</code> .
name	a human-readable name for the reference plan. Defaults to the name of <code>ref_plan</code> .

**Value**

a modified `redist_plans` object containing the reference plan

---

avg_by_prec	<i>Average a variable by precinct (Deprecated)</i>
-------------	--

---

**Description**

Deprecated in favor of `proj_avg()`. Takes a column of a `redist_plans` object and averages it across a set of draws for each precinct.

**Usage**

```
avg_by_prec(plans, x, draws = NA)
```

**Arguments**

plans	a <code>redist_plans</code> object
x	an expression to average. Tidy-evaluated in <code>plans</code> .
draws	which draws to average. <code>NULL</code> will average all draws, including reference plans. The special value <code>NA</code> will average all sampled draws. An integer, logical, or character vector indicating specific draws may also be provided.

**Value**

a vector of length matching the number of precincts, containing the average.

---

classify_plans	<i>Hierarchically classify a set of redistricting plans</i>
----------------	---

---

### Description

Applies hierarchical clustering to a distance matrix computed from a set of plans and takes the first  $k$  splits.

### Usage

```
classify_plans(dist_mat, k = 8, method = "complete")
```

### Arguments

dist_mat	a distance matrix, the output of <a href="#">plan_distances()</a>
k	the number of groupings to create
method	the clustering method to use. See <a href="#">hclust()</a> for options.

### Value

An object of class `redist_classified`, which is a list with two elements:

groups	A character vector of group labels of the form "I.A.1.a.i", one for each plan.
splits	A list of splits in the hierarchical clustering. Each list element is a list of two mutually exclusive vectors of plan indices, labeled by their group classification, indicating the plans on each side of the split.

Use [plot.redist\\_classified\(\)](#) for a visual summary.

---

compare_plans	<i>Make a comparison between two sets of plans</i>
---------------	--

---

### Description

This function provides one way to identify the structural differences between two sets of redistricting plans. It operates by computing the precinct co-occurrence matrix (a symmetric matrix where the  $i,j$ -th entry is the fraction of plans where precinct  $i$  and  $j$  are in the same district) for each set, and then computing the first eigenvalue of the difference in these two matrices (in each direction). These eigenvalues identify the important parts of the map.

**Usage**

```
compare_plans(
  plans,
  set1,
  set2,
  shp = NULL,
  plot = "fill",
  thresh = 0.1,
  labs = c("Set 1", "Set 2"),
  ncores = 1
)
```

**Arguments**

plans	a <a href="#">redist_plans</a> object
set1	<a href="#">&lt;data-masking&gt;</a> indexing vectors for the plan draws to compare. Alternatively, a second <a href="#">redist_plans</a> object to compare to.
set2	<a href="#">&lt;data-masking&gt;</a> indexing vectors for the plan draws to compare. Must be mutually exclusive with set1.
shp	a shapefile for plotting.
plot	If plot="line", display a plot for each set showing the set of boundaries which most distinguish it from the other set (the squared differences in the eigenvector values across the boundary). If plot="fill", plot the eigenvector for each set as a choropleth. If plot = 'adj', plot the shows the adjacency graph edges which most distinguish it from the other set. The adj option is a different graphical option of the same information as the line option. See below for more information. Set to FALSE to disable plotting (or leave out shp).
thresh	the value to threshold the eigenvector at in determining the relevant set of precincts for comparison.
labs	the names of the panels in the plot.
ncores	the number of parallel cores to use.

**Details**

The co-occurrence matrices are regularized with a  $Beta(1/ndists, 1 - 1/ndists)$  prior, which is useful for when either set1 or set2 is small.

**Value**

If possible, makes a comparison plot according to plot. Otherwise returns the following list:

eigen1	A numeric vector containing the first eigenvector of $p1 - p2$ , where $p1$ and $p2$ are the co-occurrence matrices for set1 and set2, respectively.
eigen2	A numeric vector containing the first eigenvector of $p2 - p1$ , where $p1$ and $p2$ are the co-occurrence matrices for set1 and set2, respectively.

group_1a, group_1b	Lists of precincts. Compared to set2, in the set1 plans these precincts were much more likely to be in separate districts. Computed by thresholding eigen1 at thresh.
group_2a, group_2b	Lists of precincts. Compared to set1, in the set2 plans these precincts were much more likely to be in separate districts. Computed by thresholding eigen2 at thresh.
cooccur_sep_1	The difference in the average co-occurrence of precincts in group_1a and group_1b between set2 and set1. Higher indicates better separation.
cooccur_sep_2	The difference in the average co-occurrence of precincts in group_2a and group_2b between set1 and set2. Higher indicates better separation.

### Examples

```
data(iowa)
iowa_map <- redist_map(iowa, ndists = 4, pop_tol = 0.05)
plans1 <- redist_smc(iowa_map, 100, silent = TRUE)
plans2 <- redist_mergesplit(iowa_map, 200, warmup = 100, silent = TRUE)
compare_plans(plans1, plans2, shp = iowa_map)
compare_plans(plans2, as.integer(draw) <= 20,
  as.integer(draw) > 20, shp = iowa_map, plot = "line")
```

---

competitiveness	<i>Compute Competitiveness</i>
-----------------	--------------------------------

---

### Description

Currently only implements the competitiveness function in equation (5) of Cho & Liu 2016.

### Usage

```
competitiveness(map, rvote, dvote, .data = cur_plans())

redist.competitiveness(plans, rvote, dvote, alpha = 1, beta = 1)
```

### Arguments

map	a <a href="#">redist_map</a> object
rvote	A numeric vector with the Republican vote for each precinct.
dvote	A numeric vector with the Democratic vote for each precinct.
.data	a <a href="#">redist_plans</a> object
plans	A numeric vector (if only one map) or matrix with one row for each precinct and one column for each map. Required.
alpha	A numeric value for the alpha parameter for the talisman metric
beta	A numeric value for the beta parameter for the talisman metric

**Value**

Numeric vector with competitiveness scores

**Examples**

```
data(f125)
data(f125_enum)

plans_05 <- f125_enum$plans[, f125_enum$pop_dev <= 0.05]
# old: comp <- redist.competitiveness(plans_05, f125$mccain, f125$obama)
comp <- compet_talisman(plans_05, f125, mccain, obama)
```

---

 constraints

*Sampling constraints*


---

**Description**

The `redist_smc()` and `redist_mergesplit()` algorithms in this package allow for additional constraints on the redistricting process to be encoded in the target distribution for sampling. These functions are provided to specify these constraints. All arguments are quoted and evaluated in the context of the data frame provided to `redist_constr()`.

**Usage**

```
add_constr_status_quo(constr, strength, current)
```

```
add_constr_grp_pow(
  constr,
  strength,
  group_pop,
  total_pop = NULL,
  tgt_group = 0.5,
  tgt_other = 0.5,
  pow = 1
)
```

```
add_constr_grp_hinge(
  constr,
  strength,
  group_pop,
  total_pop = NULL,
  tgts_group = c(0.55)
)
```

```
add_constr_grp_inv_hinge(
  constr,
```

```

    strength,
    group_pop,
    total_pop = NULL,
    tgts_group = c(0.55)
)

add_constr_compet(constr, strength, dvote, rvote, pow = 0.5)

add_constr_incumbency(constr, strength, incumbents)

add_constr_splits(constr, strength, admin)

add_constr_multisplits(constr, strength, admin)

add_constr_total_splits(constr, strength, admin)

add_constr_pop_dev(constr, strength)

add_constr_segregation(constr, strength, group_pop, total_pop = NULL)

add_constr_polsby(constr, strength, perim_df = NULL)

add_constr_fry_hold(
  constr,
  strength,
  total_pop = NULL,
  ssdmat = NULL,
  denominator = 1
)

add_constr_log_st(constr, strength, admin = NULL)

add_constr_edges_rem(constr, strength)

add_constr_custom(constr, strength, fn)

```

### Arguments

<code>constr</code>	A <code>redist_constr()</code> object
<code>strength</code>	The strength of the constraint. Higher values mean a more restrictive constraint.
<code>current</code>	The reference map for the status quo constraint.
<code>group_pop</code>	A vector of group population
<code>total_pop</code>	A vector of total population. Defaults to the population vector used for sampling.
<code>tgt_group, tgt_other</code>	Target group shares for the power-type constraint.
<code>pow</code>	The exponent for the power-type constraint.
<code>tgts_group</code>	A vector of target group shares for the hinge-type constraint.

<code>dvote, rvote</code>	A vector of Democratic or Republican vote counts
<code>incumbents</code>	A vector of unit indices for incumbents. For example, if three incumbents live in the precincts that correspond to rows 1, 2, and 100 of your <code>redist_map</code> , entering <code>incumbents = c(1, 2, 100)</code> would avoid having two or more incumbents be in the same district.
<code>admin</code>	A vector indicating administrative unit membership
<code>perim_df</code>	A dataframe output from <code>redistmetrics::prep_perims</code>
<code>ssdmat</code>	Squared distance matrix for Fryer Holden constraint
<code>denominator</code>	Fryer Holden minimum value to normalize by. Default is 1 (no normalization).
<code>fn</code>	A function

## Details

All constraints are fed into a Gibbs measure, with coefficients on each constraint set by the corresponding strength parameter. The strength can be any real number, with zero corresponding to no constraint. Higher and higher strength values will eventually cause the algorithm's accuracy and efficiency to suffer. Whenever you use constraints, be sure to check all sampling diagnostics.

The `status_quo` constraint adds a term measuring the variation of information distance between the plan and the reference, rescaled to  $[0, 1]$ .

The `grp_hinge` constraint takes a list of target group percentages. It matches each district to its nearest target percentage, and then applies a penalty of the form  $\sqrt{\max(0, tgt - group pct)}$ , summing across districts. This penalizes districts which are below their target percentage. Use `plot.redist_constr()` to visualize the effect of this constraint and calibrate strength appropriately.

The `grp_inv_hinge` constraint takes a list of target group percentages. It matches each district to its nearest target percentage, and then applies a penalty of the form  $\sqrt{\max(0, group pct - tgt)}$ , summing across districts. This penalizes districts which are above their target percentage. Use `plot.redist_constr()` to visualize the effect of this constraint and calibrate strength appropriately.

The `grp_pow` constraint (for expert use) adds a term of the form  $(|tgt_{group} - group pct| |tgt_{other} - group pct|)^{pow}$ , which encourages districts to have group shares near either `tgt_group` or `tgt_other`. Values of strength depend heavily on the values of these parameters and especially the `pow` parameter. Use `plot.redist_constr()` to visualize the effect of this constraint and calibrate strength appropriately.

The `compet` constraint encourages competitiveness by applying the `grp_pow` constraint with target percentages set to 50%. For convenience, it is specified with Democratic and Republican vote shares.

The `incumbency` constraint adds a term counting the number of districts containing paired-up incumbents. Values of strength should generally be small, given that the underlying values are counts.

The `splits` constraint adds a term counting the number of counties which are split once or more. Values of strength should generally be small, given that the underlying values are counts.

The `multisplits` constraint adds a term counting the number of counties which are split twice or more. Values of strength should generally be small, given that the underlying values are counts.

The `total_splits` constraint adds a term counting the total number of times each county is split, summed across counties (i.e., counting the number of excess district-county pairs). Values of strength should generally be small, given that the underlying values are counts.

The `edges_rem` constraint adds a term counting the number of edges removed from the adjacency graph. This is only usable with `redist_flip()`, as other algorithms implicitly use this via the compactness parameter. Values of strength should generally be small, given that the underlying values are counts.

The `log_st` constraint adds a term counting the log number of spanning trees. This is only usable with `redist_flip()`, as other algorithms implicitly use this via the compactness parameter.

The `polsby` constraint adds a term encouraging compactness as defined by the Polsby Popper metric. Values of strength may be of moderate size.

The `fry_hold` constraint adds a term encouraging compactness as defined by the Fryer Holden metric. Values of strength should be extremely small, as the underlying values are massive when the true minimum Fryer Holden denominator is not known.

The `segregation` constraint adds a term encouraging segregation among minority groups, as measured by the dissimilarity index.

The `pop_dev` constraint adds a term encouraging plans to have smaller population deviations from the target population.

The `custom` constraint allows the user to specify their own constraint using a function which evaluates districts one at a time. The provided function `fn` should take two arguments: a vector describing the current plan assignment for each unit as its first argument, and an integer describing the district which to evaluate in the second argument. `which([plans == distr])` would give the indices of the units that are assigned to a district `distr` in any iteration. The function must return a single scalar for each plan - district combination, where a value of 0 indicates no penalty is applied. If users want to penalize an entire plan, they can have the penalty function return a scalar that does not depend on the district. It is important that `fn` not use information from precincts not included in `distr`, since in the case of SMC these precincts may not be assigned any district at all (`plan` will take the value of 0 for these precincts). The flexibility of this constraint comes with an additional computational cost, since the other constraints are written in C++ and so are more performant.

## Examples

```
data(iowa)
iowa_map <- redist_map(iowa, existing_plan = cd_2010, pop_tol = 0.05)
constr <- redist_constr(iowa_map)
constr <- add_constr_splits(constr, strength = 1.5, admin = name)
constr <- add_constr_grp_hinge(constr, strength = 100,
  dem_08, tot_08, tgts_group = c(0.5, 0.6))
# encourage districts to have the same number of counties
constr <- add_constr_custom(constr, strength = 1000, fn = function(plan, distr) {
  # notice that we only use information on precincts in `distr`
  abs(sum(plan == distr) - 99/4)
})
print(constr)
```

---

county_splits	<i>Count County Splits</i>
---------------	----------------------------

---

**Description**

Count County Splits

**Usage**

```
county_splits(map, counties, .data = cur_plans())

redist_splits(plans, counties)
```

**Arguments**

map	a <a href="#">redist_map</a> object
counties	A vector of county names or county ids.
.data	a <a href="#">redist_plans</a> object
plans	A numeric vector (if only one map) or matrix with one row for each precinct and one column for each map. Required.

**Value**

integer vector with one number for each map

---

distr_compactness	<i>Calculate compactness measures for a set of plans</i>
-------------------	--

---

**Description**

`redist.compactness` is used to compute different compactness statistics for a shapefile. It currently computes the Polsby-Popper, Schwartzberg score, Length-Width Ratio, Convex Hull score, Reock score, Boyce Clark Index, Fryer Holden score, Edges Removed number, and the log of the Spanning Trees.

**Usage**

```
distr_compactness(map, measure = "FracKept", .data = cur_plans(), ...)

redist.compactness(
  shp = NULL,
  plans,
  measure = c("PolsbyPopper"),
  total_pop = NULL,
  adj = NULL,
```

```

draw = 1,
ncores = 1,
counties = NULL,
planarize = 3857,
ppRcpp,
perim_path,
perim_df
)

```

### Arguments

<code>map</code>	a <a href="#">redist_map</a> object
<code>measure</code>	A vector with a string for each measure desired. "PolsbyPopper", "Schwartzberg", "LengthWidth", "ConvexHull", "Reock", "BoyceClark", "FryerHolden", "EdgesRemoved", "FracKept", and "logSpanningTree" are implemented. Defaults to "PolsbyPopper". Use "all" to return all implemented measures.
<code>.data</code>	a <a href="#">redist_plans</a> object
<code>...</code>	passed on to <code>redist.compactness</code>
<code>shp</code>	A <code>SpatialPolygonsDataFrame</code> or <code>sf</code> object. Required unless "EdgesRemoved" and "logSpanningTree" with adjacency provided.
<code>plans</code>	A numeric vector (if only one map) or matrix with one row for each precinct and one column for each map. Required.
<code>total_pop</code>	A numeric vector with the population for every observation. Is only necessary when "FryerHolden" is used for measure. Defaults to NULL.
<code>adj</code>	A zero-indexed adjacency list. Only used for "PolsbyPopper", "EdgesRemoved" and "logSpanningTree". Created with <code>redist.adjacency</code> if not supplied and needed. Default is NULL.
<code>draw</code>	A numeric to specify draw number. Defaults to 1 if only one map provided and the column number if multiple maps given. Can also take a factor input, which will become the draw column in the output if its length matches the number of entries in plans. If the plans input is a <code>redist_plans</code> object, it extracts the draw identifier.
<code>ncores</code>	Number of cores to use for parallel computing. Default is 1.
<code>counties</code>	A numeric vector from 1:ncounties corresponding to counties. Required for "logSpanningTree".
<code>planarize</code>	a number, indicating the CRS to project the shapefile to if it is latitude-longitude based. Set to FALSE to avoid planarizing.
<code>ppRcpp</code>	Boolean, whether to run Polsby Popper and Schwartzberg using Rcpp. It has a higher upfront cost, but quickly becomes faster. Becomes TRUE if <code>ncol(district_membership) &gt; 8</code> and not manually set.
<code>perim_path</code>	it checks for an Rds, if no rds exists at the path, it creates an rds with borders and saves it. This can be created in advance with <code>redistmetrics::prep_perims()</code> .
<code>perim_df</code>	A dataframe output from <code>redistmetrics::prep_perims()</code> .

## Details

This function computes specified compactness scores for a map. If there is more than one shape specified for a single district, it combines them, if necessary, and computes one score for each district.

Polsby-Popper is computed as

$$\frac{4 * \pi * A(d)}{P(d)^2}$$

where A is the area function, the district is d, and P is the perimeter function. All values are between 0 and 1, where larger values are more compact.

Schwartzberg is computed as

$$\frac{P(d)}{2 * \pi * \sqrt{\frac{A(d)}{\pi}}}$$

where A is the area function, the district is d, and P is the perimeter function. All values are between 0 and 1, where larger values are more compact.

The Length Width ratio is computed as

$$\frac{length}{width}$$

where length is the shorter of the maximum x distance and the maximum y distance. Width is the longer of the two values. All values are between 0 and 1, where larger values are more compact.

The Convex Hull score is computed as

$$\frac{A(d)}{A(CVH)}$$

where A is the area function, d is the district, and CVH is the convex hull of the district. All values are between 0 and 1, where larger values are more compact.

The Reock score is computed as

$$\frac{A(d)}{A(MBC)}$$

where A is the area function, d is the district, and MBC is the minimum bounding circle of the district. All values are between 0 and 1, where larger values are more compact.

The Boyce Clark Index is computed as

$$1 - \sum_1^{16} \left\{ \frac{|\frac{r_i}{\sum_i r_i} * 100 - 6.25|}{200} \right\}$$

. The  $r_i$  are the distances of the 16 radii computed from the geometric centroid of the shape to the most outward point of the shape that intersects the radii, if the centroid is contained within the shape. If the centroid lies outside of the shape, a point on the surface is used, which will naturally incur a penalty to the score. All values are between 0 and 1, where larger values are more compact.

The Fryer Holden score for each district is computed with

$$Pop \odot D(precinct)^2$$

, where  $Pop$  is the population product matrix. Each element is the product of the i-th and j-th precinct's populations.  $D$  represents the distance, where the matrix is the distance between each

precinct. To fully compute this index, for any map, the sum of these values should be used as the numerator. The denominator can be calculated from the full enumeration of districts as the smallest calculated numerator. This produces very large numbers, where smaller values are more compact.

The log spanning tree measure is the logarithm of the product of the number of spanning trees which can be drawn on each district.

The edges removed measure is number of edges removed from the underlying adjacency graph. A smaller number of edges removed is more compact.

The fraction kept measure is the fraction of edges that were not removed from the underlying adjacency graph. This takes values 0 - 1, where 1 is more compact.

### Value

A tibble with a column that specifies the district, a column for each specified measure, and a column that specifies the map number.

### References

- Boyce, R., & Clark, W. 1964. The Concept of Shape in Geography. *Geographical Review*, 54(4), 561-572.
- Cox, E. 1927. A Method of Assigning Numerical and Percentage Values to the Degree of Roundness of Sand Grains. *Journal of Paleontology*, 1(3), 179-183.
- Fryer R, Holden R. 2011. Measuring the Compactness of Political Districting Plans. *Journal of Law and Economics*.
- Harris, Curtis C. 1964. "A scientific method of districting". *Behavioral Science* 3(9), 219-225.
- Maceachren, A. 1985. Compactness of Geographic Shape: Comparison and Evaluation of Measures. *Geografiska Annaler. Series B, Human Geography*, 67(1), 53-67.
- Polsby, Daniel D., and Robert D. Popper. 1991. "The Third Criterion: Compactness as a procedural safeguard against partisan gerrymandering." *Yale Law & Policy Review* 9 (2): 301-353.
- Reock, E. 1961. A Note: Measuring Compactness as a Requirement of Legislative Apportionment. *Midwest Journal of Political Science*, 5(1), 70-74.
- Schwartzberg, Joseph E. 1966. Reapportionment, Gerrymanders, and the Notion of Compactness. *Minnesota Law Review*. 1701.

### Examples

```
data(f125)
data(f125_enum)

plans_05 <- f125_enum$plans[, f125_enum$pop_dev <= 0.05]

# old redist.compactness(
#   shp = f125, plans = plans_05[, 1:3],
#   measure = c("PolsbyPopper", "EdgesRemoved")
# )
comp_polsby(plans_05[, 1:3], f125)
comp_edges_rem(plans_05[, 1:3], f125, f125$adj)
```

---

EPSG

*EPSG Table*

---

### Description

This data contains NAD83 (HARN) EPSG codes for every U.S. state. Since `redist` uses projected geometries, it is often a good idea to use projections tailored to a particular state, rather than, for example, a Mercator projection. Use these codes along with `sf::st_transform()` to project your shapefiles nicely.

### Usage

```
data("EPSG")
```

### Format

named list containing EPSG codes for each U.S. state. Codes are indexed by state abbreviations.

### Examples

```
data(EPSG)
EPSG$WA # 2855
```

---

f125

*Florida 25 Precinct Shape File*

---

### Description

This data set contains the 25-precinct shapefile and related data for each precinct. All possible partitions of the 25 precincts into three contiguous congressional districts are stored in `f125_enum`, and the corresponding adjacency graph is stored in `f125_adj`. This is generally useful for demonstrating basic algorithms locally.

### Usage

```
data("f125")
```

### Format

`sf` data.frame containing columns for useful data related to the redistricting process, subsetted from real data in Florida, and `sf` geometry column.

`geoid` Contains unique identifier for each precinct which can be matched to the full Florida dataset.

`pop` Contains the population of each precinct.

`vap` Contains the voting age population of each precinct.

`obama` Contains the 2012 presidential vote for Obama.

mccain Contains the 2012 presidential vote for McCain.  
 TotPop Contains the population of each precinct. Identical to pop.  
 BlackPop Contains the black population of each precinct.  
 HispPop Contains the Hispanic population of each precinct.  
 VAP Contains the voting age population of each precinct. Identical to vap.  
 BlackVAP Contains the voting age population of black constituents of each precinct.  
 HispVAP Contains the voting age population of hispanic constituents of each precinct.  
 geometry Contains sf geometry of each precinct.

## References

Fifield, Benjamin, Michael Higgins, Kosuke Imai and Alexander Tarr. (2016) "A New Automated Redistricting Simulator Using Markov Chain Monte Carlo." Working Paper. Available at <http://imai.princeton.edu/research/files/redist.pdf>.

## Examples

```
data(fl25)
```

---

fl250

*Florida 250 Precinct Shape File*

---

## Description

This data set contains the 250 Precinct shapefile and related data for each precinct.

## Usage

```
data("fl250")
```

## Format

sf data.frame containing columns for useful data related to the redistricting process, subsetted from real data in Florida, and sf geometry column.

geoid Contains unique identifier for each precinct which can be matched to the full Florida dataset.

pop Contains the population of each precinct.

vap Contains the voting age population of each precinct.

obama Contains the 2012 presidential vote for Obama.

mccain Contains the 2012 presidential vote for McCain.

TotPop Contains the population of each precinct. Identical to pop.

BlackPop Contains the black population of each precinct.

HispPop Contains the Hispanic population of each precinct.

VAP Contains the voting age population of each precinct. Identical to vap.  
 BlackVAP Contains the voting age population of black constituents of each precinct.  
 HispVAP Contains the voting age population of hispanic constituents of each precinct.  
 geometry Contains sf geometry of each precinct.

### Details

It is a random 70 precinct connected subset from Florida's precincts. This was introduced by [doi:10.1080/2330443X.2020.1791773](https://doi.org/10.1080/2330443X.2020.1791773)

### References

Benjamin Fifield, Kosuke Imai, Jun Kawahara & Christopher T. Kenny (2020) The Essential Role of Empirical Validation in Legislative Redistricting Simulation, *Statistics and Public Policy*, 7:1, 52-68, [doi:10.1080/2330443X.2020.1791773](https://doi.org/10.1080/2330443X.2020.1791773)

### Examples

```
data(fl250)
```

---

fl25_adj	<i>Florida 25 Precinct File</i>
----------	---------------------------------

---

### Description

This data set contains the 25-precinct shapefile and related data for each precinct. All possible partitions of the 25 precincts into three contiguous congressional districts are stored in [fl25\\_enum](#), and the corresponding adjacency graph is stored in [fl25\\_adj](#).

### Format

A list storing the adjacency graph for the 25-precinct subset of Florida.

### References

Fifield, Benjamin, Michael Higgins, Kosuke Imai and Alexander Tarr. (2016) "A New Automated Redistricting Simulator Using Markov Chain Monte Carlo." Working Paper. Available at <http://imai.princeton.edu/research/files/redist.pdf>.

### Examples

```
data(fl25_adj)
```

---

f125_enum	<i>All Partitions of 25 Precincts into 3 Congressional Districts (No Population Constraint)</i>
-----------	---

---

**Description**

This data set contains demographic and geographic information about 25 contiguous precincts in the state of Florida. The data lists all possible partitions of the 25 precincts into three contiguous congressional districts. The 25-precinct shapefile may be found in [f125](#)

**Usage**

```
data("f125_enum")
```

**Format**

A list with two entries:

`plans` A matrix containing every partition of the 25 precincts into three contiguous congressional districts, with no population constraint.

`pop_dev` A vector containing the maximum population deviation across the three districts for each plan.

**References**

Fifield, Benjamin, Michael Higgins, Kosuke Imai and Alexander Tarr. (2016) "A New Automated Redistricting Simulator Using Markov Chain Monte Carlo." Working Paper. Available at <http://imai.princeton.edu/research/files/redist.pdf>.

Massey, Douglas and Nancy Denton. (1987) "The Dimensions of Social Segregation". Social Forces.

**Examples**

```
data(f125_enum)
```

---

f170	<i>Florida 70 Precinct Shape File</i>
------	---------------------------------------

---

**Description**

This data set contains the 70 Precinct shapefile and related data for each precinct.

**Usage**

```
data("f170")
```

**Format**

sf data.frame containing columns for useful data related to the redistricting process, subsetted from real data in Florida, and sf geometry column.

geoid Contains unique identifier for each precinct which can be matched to the full Florida dataset.

pop Contains the population of each precinct.

vap Contains the voting age population of each precinct.

obama Contains the 2012 presidential vote for Obama.

mccain Contains the 2012 presidential vote for McCain.

TotPop Contains the population of each precinct. Identical to pop.

BlackPop Contains the black population of each precinct.

HispPop Contains the Hispanic population of each precinct.

VAP Contains the voting age population of each precinct. Identical to vap.

BlackVAP Contains the voting age population of black constituents of each precinct.

HispVAP Contains the voting age population of hispanic constituents of each precinct.

geometry Contains sf geometry of each precinct.

**Details**

It is a random 70 precinct connected subset from Florida's precincts. This was introduced by [doi:10.1080/2330443X.2020.1791773](https://doi.org/10.1080/2330443X.2020.1791773)

**References**

Benjamin Fifield, Kosuke Imai, Jun Kawahara & Christopher T. Kenny (2020) The Essential Role of Empirical Validation in Legislative Redistricting Simulation, *Statistics and Public Policy*, 7:1, 52-68, doi:10.1080/2330443X.2020.1791773

**Examples**

```
data(f170)
```

---

freeze	<i>Freeze Parts of a Map</i>
--------	------------------------------

---

**Description**

Freeze Parts of a Map

**Usage**

```
freeze(freeze_row, plan, .data = cur_map())
```

```
redist.freeze(adj, freeze_row, plan = rep(1, length(adj)))
```

**Arguments**

freeze_row	Required, logical vector where TRUE freezes and FALSE lets a precinct stay free or a vector of indices to freeze
plan	A vector of district assignments, which if provided will create separate groups by district. Recommended. In freeze defaults to the existing plan, if one exists.
.data	a <a href="#">redist_map</a> object
adj	Required, zero indexed adjacency list.

**Value**

integer vector to group by

**Examples**

```
library(redist)
library(dplyr)
data(fl25)
data(fl25_enum)
data(fl25_adj)
plan <- fl25_enum$plans[, 5118]
freeze_id <- redist.freeze(adj = fl25_adj, freeze_row = (plan == 2),
  plan = plan)

data(iowa)
map <- redist_map(iowa, existing_plan = cd_2010, pop_tol = 0.02)
map <- map %>% merge_by(freeze(cd_2010 == 1, .data = .))
```

---

get\_adj

*Get and set the adjacency graph from a redist\_map object*


---

**Description**

Get and set the adjacency graph from a redist\_map object

**Usage**

```
get_adj(x)

set_adj(x, adj)
```

**Arguments**

x	the redist_map object
adj	a new adjacency list.

**Value**

a zero-indexed adjacency list (get\_adj)  
the modified redist\_map object (set\_adj)

---

get_existing	<i>Extract the existing district assignment from a redist_map object</i>
--------------	--

---

**Description**

Extract the existing district assignment from a redist\_map object

**Usage**

get\_existing(x)

**Arguments**

x                    the redist\_map object

**Value**

an integer vector of district numbers

---

get_mh_acceptance_rate	<i>Extract the Metropolis Hastings Acceptance Rate</i>
------------------------	--

---

**Description**

Extract the Metropolis Hastings Acceptance Rate

**Usage**

get\_mh\_acceptance\_rate(plans)

**Arguments**

plans                the redist\_plans object

**Value**

a numeric acceptance rate

---

get_plans_matrix	<i>Extract the matrix of district assignments from a redistricting simulation</i>
------------------	---

---

**Description**

Extract the matrix of district assignments from a redistricting simulation

**Usage**

```
get_plans_matrix(x)

## S3 method for class 'redist_plans'
as.matrix(x, ...)
```

**Arguments**

x	the redist_plans object
...	ignored

**Value**

matrix  
matrix

---

get_plans_weights	<i>Extract the sampling weights from a redistricting simulation.</i>
-------------------	--

---

**Description**

May be NULL if no weights exist (MCMC or optimization methods).

**Usage**

```
get_plans_weights(plans)

## S3 method for class 'redist_plans'
weights(object, ...)
```

**Arguments**

plans, object	the redist_plans object
...	Ignored.

**Value**

A numeric vector of weights, with an additional attribute `resampled` indicating whether the plans have been resampled according to these weights. If weights have been resampled, this returns the weights before resampling (i.e., they do not correspond to the resampled plans).

numeric vector

---

get_pop_tol	<i>Get and set the population tolerance from a <code>redist_map</code> object</i>
-------------	---

---

**Description**

Get and set the population tolerance from a `redist_map` object

**Usage**

```
get_pop_tol(map)
```

```
set_pop_tol(map, pop_tol)
```

**Arguments**

map	the <code>redist_map</code> object
pop_tol	the population tolerance

**Value**

For `get_pop_tol`, a single numeric value, the population tolerance

For `set_pop_tol`, an updated `redist_map` object

---

get_sampling_info	<i>Extract the sampling information from a redistricting simulation</i>
-------------------	---

---

**Description**

Extract the sampling information from a redistricting simulation

**Usage**

```
get_sampling_info(plans)
```

**Arguments**

plans	the <code>redist_plans</code> object
-------	--------------------------------------

**Value**

a list of parameters and information about the sampling problem.

---

get_target	<i>Extract the target district population from a redist_map object</i>
------------	--

---

**Description**

Extract the target district population from a redist\_map object

**Usage**

```
get_target(x)
```

**Arguments**

x                    the redist\_map object

**Value**

a single numeric value, the target population

---

group_frac	<i>Calculate Group Proportion by District</i>
------------	---

---

**Description**

redist.group.percent computes the proportion that a group makes up in each district across a matrix of maps.

**Usage**

```
group_frac(
  map,
  group_pop,
  total_pop = map[[attr(map, "pop_col")]],
  .data = pl()
)
```

```
redist.group.percent(plans, group_pop, total_pop, ncores = 1)
```

**Arguments**

map	a <a href="#">redist_map</a> object
group_pop	A numeric vector with the population of the group for every precinct.
total_pop	A numeric vector with the population for every precinct.
.data	a <a href="#">redist_plans</a> object or matrix of plans
plans	A matrix with one row for each precinct and one column for each map. Required.
ncores	Number of cores to use for parallel computing. Default is 1.

**Value**

matrix with percent for each district

**Examples**

```
data(f125)
data(f125_enum)

cd <- f125_enum$plans[, f125_enum$pop_dev <= 0.05]
f125_map = redist_map(f125, ndists=3, pop_tol=0.1)
f125_plans = redist_plans(cd, f125_map, algorithm="enumpart")

group_frac(f125_map, BlackPop, TotPop, f125_plans)
```

---

iowa

*Iowa County File*


---

**Description**

This data contains geographic and demographic information on the 99 counties of the state of Iowa.

**Usage**

```
data("iowa")
```

**Format**

sf tibble containing columns for useful data related to the redistricting process

fips The FIPS code for the county.

cd\_2010 The 2010 congressional district assignments.

pop The total population of the precinct, according to the 2010 Census.

white The non-Hispanic white population of the precinct.

black The non-Hispanic Black population of the precinct.

hisp The Hispanic population (of any race) of the precinct.

vap The voting-age population of the precinct.

wvap The white voting-age population of the precinct.

bvap The Black voting-age population of the precinct.

hvap The Hispanic voting-age population of the precinct.

tot\_08 Number of total votes for president in the county in 2008.

dem\_08 Number of votes for Barack Obama in 2008.

rep\_08 Number of votes for John McCain in 2008.

region The 28E agency regions for counties.

geometry The sf geometry column containing the geographic information.

**Examples**

```
data(iowa)
print(iowa)
```

---

is_contiguous	<i>Check that a redist_map object is contiguous</i>
---------------	---

---

**Description**

Check that a redist\_map object is contiguous

**Usage**

```
is_contiguous(x)
```

**Arguments**

x                    the object

**Value**

TRUE if contiguous.

---

is_county_split	<i>Identify which counties are split by a plan</i>
-----------------	--

---

**Description**

Identify which counties are split by a plan

**Usage**

```
is_county_split(plan, counties)
```

**Arguments**

plan                    A vector of precinct/unit assignments  
 counties                A vector of county names or county ids.

**Value**

A logical vector which is TRUE for precincts belonging to counties which are split

---

last_plan	<i>Extract the last plan from a set of plans</i>
-----------	--

---

**Description**

Extract the last plan from a set of plans

**Usage**

```
last_plan(plans)
```

**Arguments**

plans            A `redist_plans` object

**Value**

An integer vector containing the final plan assignment.

---

make_cores	<i>Identify Cores of a District (Heuristic)</i>
------------	---

---

**Description**

Creates a grouping ID to unite geographies and perform analysis on a smaller set of precincts. It identifies all precincts more than boundary edges of a district district boundary. Each contiguous group of precincts more than boundary steps away from another district gets it own group. Some districts may have multiple, disconnected components that make up the core, but each of these is assigned a separate grouping id so that a call to `sf::st_union()` would produce only connected pieces.

**Usage**

```
make_cores(.data = cur_map(), boundary = 1, focus = NULL)
```

```
redist.identify.cores(adj, plan, boundary = 1, focus = NULL, simplify = TRUE)
```

**Arguments**

.data            a `redist_map` object

boundary        Number of steps to check for. Defaults to 1.

focus          Optional. Integer. A single district to focus on.

adj             zero indexed adjacency list.

plan            An integer vector or matrix column of district assignments.

simplify        Optional. Logical. Whether to return extra information or just grouping ID.

## Details

This is a loose interpretation of the NCSL's summary of redistricting criteria to preserve the cores of prior districts. Using the adjacency graph for a given plan, it will locate the precincts on the boundary of the district, within boundary steps of the edge. Each of these is given their own group. Each remaining entry that is not near the boundary of the district is given an id that can be used to group the remainder of the district by connected component. This portion is deemed the core of the district.

## Value

integer vector (if simplify is false). Otherwise it returns a tibble with the grouping variable as group\_id and additional information on connected components.

## See Also

[redist.plot.cores\(\)](#) for a plotting function

## Examples

```
data(f1250)
f1250_map <- redist_map(f1250, ndists = 4, pop_tol = 0.01)
plan <- as.matrix(redist_smc(f1250_map, 20, silent = TRUE))
core <- redist.identify.cores(adj = f1250_map$adj, plan = plan)
redist.plot.cores(shp = f1250, plan = plan, core = core)
```

---

match\_numbers

*Renumber districts to match an existing plan*

---

## Description

District numbers in simulated plans are by and large random. This function attempts to renumber the districts across all simulated plans to match the numbers in a provided plan, using the Hungarian algorithm.

## Usage

```
match_numbers(  
  data,  
  plan,  
  total_pop = attr(data, "prec_pop"),  
  col = "pop_overlap"  
)
```

**Arguments**

data	a <code>redist_plans</code> object.
plan	a character vector giving the name of the plan to match to (e.g., for a reference plan), or an integer vector containing the plan itself.
total_pop	a vector of population counts. Should not be needed for most <code>redist_plans</code> objects.
col	the name of a new column to store the vector of population overlap with the reference plan: the fraction of the total population who are in the same district under each plan and the reference plan. Set to <code>NULL</code> if no column should be created. renumbering options in any plan.

**Value**

a modified `redist_plans` object. New district numbers will be stored as an ordered factor variable in the `district` column. The district numbers in the plan matrix will match the levels of this factor.

**Examples**

```
data(iowa)

iowa_map <- redist_map(iowa, existing_plan = cd_2010, pop_tol = 0.05)
plans <- redist_smc(iowa_map, 100, silent = TRUE)
match_numbers(plans, "cd_2010")
```

---

merge\_by

---

*Merge map units*


---

**Description**

In performing a county-level or cores-based analysis it is often necessary to merge several units together into a larger unit. This function performs this operation, modifying the adjacency graph as needed and attempting to properly aggregate other data columns.

**Usage**

```
merge_by(.data, ..., by_existing = TRUE, drop_geom = TRUE, collapse_chr = TRUE)
```

**Arguments**

<code>.data</code>	a <code>redist_map</code> object
<code>...</code>	<code>&lt;tidy-select&gt;</code> the column(s) to merge by
<code>by_existing</code>	if an existing assignment is present, whether to also group by it
<code>drop_geom</code>	whether to drop the geometry column. Recommended, as otherwise a costly geometric merge is required.
<code>collapse_chr</code>	if <code>TRUE</code> , preserve character columns by collapsing their values. For example, a county name column in Iowa might be merged and have entries such as "Cedar~Clinton~Des Moines". Set to <code>FALSE</code> to drop character columns instead.

**Value**

A merged `redist_map` object

---

min_move_parity	<i>Calculates Sparse Population Moves to Minimize Population Deviation</i>
-----------------	--

---

**Description**

This function computes a minimal set of population moves (e.g., 5 people from district 1 to district 3) to maximally balance the population between districts. The moves are only allowed between districts that share the territory of a county, so that any boundary adjustments are guaranteed to preserve all unbroken county boundaries.

**Usage**

```
min_move_parity(map, plan, counties = NULL, penalty = 0.2)
```

**Arguments**

map	a <code>redist_map</code>
plan	an integer vector containing the plan to be balanced. Tidy-evaluated.
counties	an optional vector of counties, whose boundaries will be preserved. Tidy-evaluated.
penalty	the larger this value, the more to encourage sparsity.

**Value**

a list with components:

`moves` A tibble describing the population moves

`pop_old` The current district populations

`pop_new` The district populations after the moves

**Examples**

```
data(iowa)
iowa_map <- redist_map(iowa, existing_plan = cd_2010, pop_tol = 0.01)
min_move_parity(iowa_map, cd_2010)
```

---

muni_splits	<i>Counts the Number of Municipalities Split Between Districts</i>
-------------	--

---

**Description**

Counts the total number of municipalities that are split. Municipalities in this interpretation do not need to cover the entire state, which differs from counties.

**Usage**

```
muni_splits(map, munis, .data = cur_plans())

redist.muni.splits(plans, munis)
```

**Arguments**

map	a <a href="#">redist_map</a> object
munis	A vector of municipality names or ids.
.data	a <a href="#">redist_plans</a> object
plans	A numeric vector (if only one map) or matrix with one row for each precinct and one column for each map. Required.

**Value**

integer vector of length ndist by ncol(plans)

**Examples**

```
data(iowa)
ia <- redist_map(iowa, existing_plan = cd_2010, total_pop = pop, pop_tol = 0.01)
plans <- redist_smc(ia, 50, silent = TRUE)
ia$region[1:10] <- NA
#old redist.muni.splits(plans, ia$region)
splits_sub_admin(plans, ia, region)
```

---

number_by	<i>Renumber districts to match a quantity of interest</i>
-----------	---

---

**Description**

District numbers in simulated plans are by and large random. This function will renumber the districts across all simulated plans in order of a provided quantity of interest.

**Usage**

```
number_by(data, x, desc = FALSE)
```

**Arguments**

data	a <code>redist_plans</code> object
x	<data-masking> the quantity of interest.
desc	TRUE if district should be sorted in descending order.

**Value**

a modified `redist_plans` object. New district numbers will be stored as an ordered factor variable in the `district` column. The district numbers in the plan matrix will match the levels of this factor.

---

<code>partisan_metrics</code>	<i>Calculate gerrymandering metrics for a set of plans</i>
-------------------------------	--

---

**Description**

`redist.metrics` is used to compute different gerrymandering metrics for a set of maps.

**Usage**

```
partisan_metrics(map, measure, rvote, dvote, ..., .data = cur_plans())
```

```
redist.metrics(
  plans,
  measure = "DSeats",
  rvote,
  dvote,
  tau = 1,
  biasV = 0.5,
  respV = 0.5,
  bandwidth = 0.01,
  draw = 1
)
```

**Arguments**

map	a <code>redist_map</code> object
measure	A vector with a string for each measure desired from list "DSeats", "DVS", "EffGap", "EffGapEqPop", "TauGap", "MeanMedian", "Bias", "BiasV", "Declination", "Responsiveness", "LopsidedWins", "RankedMarginal", and "SmoothedSeat". Use "all" to get all metrics. "DSeats" and "DVS" are always computed, so it is recommended to always return those values.
rvote	A numeric vector with the Republican vote for each precinct.
dvote	A numeric vector with the Democratic vote for each precinct.
...	passed on to <code>redist.metrics</code>
.data	a <code>redist_plans</code> object

plans	A numeric vector (if only one map) or matrix with one row for each precinct and one column for each map. Required.
tau	A non-negative number for calculating Tau Gap. Only used with option "Tau-Gap". Defaults to 1.
biasV	A value between 0 and 1 to compute bias at. Only used with option "BiasV". Defaults to 0.5.
respV	A value between 0 and 1 to compute responsiveness at. Only used with option "Responsiveness". Defaults to 0.5.
bandwidth	A value between 0 and 1 for computing responsiveness. Only used with option "Responsiveness." Defaults to 0.01.
draw	A numeric to specify draw number. Defaults to 1 if only one map provided and the column number if multiple maps given. Can also take a factor input, which will become the draw column in the output if its length matches the number of entries in plans. If the plans input is a redist_plans object, it extracts the draw identifier.

### Details

This function computes specified compactness scores for a map. If there is more than one precinct specified for a map, it aggregates to the district level and computes one score.

- DSeats is computed as the expected number of Democratic seats with no change in votes.
- DVS is the Democratic Vote Share, which is the two party vote share with Democratic votes as the numerator.
- EffGap is the Efficiency Gap, calculated with votes directly.
- EffGapEqPop is the Efficiency Gap under an Equal Population assumption, calculated with the DVS.
- TauGap is the Tau Gap, computed with the Equal Population assumption.
- MeanMedian is the Mean Median difference.
- Bias is the Partisan Bias computed at 0.5.
- BiasV is the Partisan Bias computed at value V.
- Declination is the value of declination at 0.5.
- Responsiveness is the responsiveness at the user-supplied value with the user-supplied bandwidth.
- LopsidedWins computed the Lopsided Outcomes value, but does not produce a test statistic.
- RankedMarginal computes the Ranked Marginal Deviation (0-1, smaller is better). This is also known as the "Gerrymandering Index" and is sometimes presented as this value divided by 10000.
- SmoothedSeat computes the Smoothed Seat Count Deviation (0-1, smaller is R Bias, bigger is D Bias).

### Value

A tibble with a column for each specified measure and a column that specifies the map number.

## References

Jonathan N. Katz, Gary King, and Elizabeth Rosenblatt. 2020. Theoretical Foundations and Empirical Evaluations of Partisan Fairness in District-Based Democracies. *American Political Science Review*, 114, 1, Pp. 164-178.

Gregory S. Warrington. 2018. "Quantifying Gerrymandering Using the Vote Distribution." *Election Law Journal: Rules, Politics, and Policy*. Pp. 39-57. <http://doi.org/10.1089/ej.2017.0447>

Samuel S.-H. Wang. 2016. "Three Tests for Practical Evaluation of Partisan Gerrymandering." *Stanford Law Review*, 68, Pp. 1263 - 1321.

Gregory Herschlag, Han Sung Kang, Justin Luo, Christy Vaughn Graves, Sachet Bangia, Robert Ravier & Jonathan C. Mattingly (2020) Quantifying Gerrymandering in North Carolina, *Statistics and Public Policy*, 7:1, 30-38, DOI: 10.1080/2330443X.2020.1796400

## Examples

```
data(f125)
data(f125_enum)
plans_05 <- f125_enum$plans[, f125_enum$pop_dev <= 0.05]
# old: redist.metrics(plans_05, measure = "DSeats", rvote = f125$mccain, dvote = f125$obama)
part_dseats(plans_05, f125, mccain, obama)
```

---

pl

*Access the Current redist\_plans() Object*

---

## Description

Useful inside piped expressions and dplyr functions.

## Usage

```
pl()
```

## Value

A redist\_plans object, or NULL if not called from inside a dplyr function.

## Examples

```
pl()
```

---

plans\_diversity      *Calculate the diversity of a set of plans*

---

### Description

Returns the off-diagonal elements of the variation of information distance matrix for a sample of plans, which can be used as a diagnostic measure to assess the diversity of a set of plans. While the exact scale varies depending on the number of precincts and districts, generally diversity is good if most of the values are greater than 0.5. Conversely, if there are many values close to zero, then the sample has many similar plans and may not be a good approximation to the target distribution.

### Usage

```
plans_diversity(  
  plans,  
  chains = 1,  
  n_max = 100,  
  ncores = 1,  
  total_pop = attr(plans, "prec_pop")  
)
```

### Arguments

plans	a <code>redist_plans</code> object.
chains	For plans objects with multiple chains, which ones to compute diversity for. Defaults to the first. Specify "all" to use all chains.
n_max	the maximum number of plans to sample in computing the distances. Larger numbers will have less sampling error but will require more computation time.
ncores	the number of cores to use in computing the distances.
total_pop	The vector of precinct populations. Used only if computing variation of information. If not provided, equal population of precincts will be assumed, i.e. the VI will be computed with respect to the precincts themselves, and not the population.

### Value

A numeric vector of off-diagonal variation of information distances.

### Examples

```
data(iowa)  
ia <- redist_map(iowa, existing_plan = cd_2010, pop_tol = 0.01)  
plans <- redist_smc(ia, 100, silent = TRUE)  
hist(plans_diversity(plans))
```

---

plan\_distances      *Compute Distance between Partitions*

---

### Description

Compute Distance between Partitions

### Usage

```
plan_distances(plans, measure = "variation of information", ncores = 1)
redist.distances(plans, measure = "Hamming", ncores = 1, total_pop = NULL)
```

### Arguments

plans	A matrix with one row for each precinct and one column for each map. Required.
measure	String vector indicating which distances to compute. Implemented currently are "Hamming", "Manhattan", "Euclidean", and "variation of information", Use "all" to return all implemented measures. Not case sensitive, and any unique substring is enough, e.g. "ham" for Hamming, or "info" for variation of information.
ncores	Number of cores to use for parallel computing. Default is 1.
total_pop	The vector of precinct populations. Used only if computing variation of information. If not provided, equal population of precincts will be assumed, i.e. the VI will be computed with respect to the precincts themselves, and not the population.

### Details

Hamming distance measures the number of different precinct assignments between plans. Manhattan and Euclidean distances are the 1- and 2-norms for the assignment vectors. All three of the Hamming, Manhattan, and Euclidean distances implemented here are not invariant to permutations of the district labels; permuting will cause large changes in measured distance, and maps which are identical up to a permutation may be computed to be maximally distant.

Variation of Information is a metric on population partitions (i.e., districtings) which is invariant to permutations of the district labels, and arises out of information theory. It is calculated as

$$VI(\xi, \xi') = - \sum_{i=1}^n \sum_{j=1}^n \text{pop}(\xi_i \cap \xi'_j) / P (2 \log(\text{pop}(\xi_i \cap \xi'_j)) - \log(\text{pop}(\xi_i)) - \log(\text{pop}(\xi'_j)))$$

where  $\xi, \xi'$  are the partitions,  $\xi_i, \xi_j$  the individual districts,  $\text{pop}(\cdot)$  is the population, and  $P$  the total population of the state. VI is also expressible as the difference between the joint entropy and the mutual information (see references).

**Value**

distance\_matrix returns a numeric distance matrix for the chosen metric.  
 a named list of distance matrices, one for each distance measure selected.

**References**

Cover, T. M. and Thomas, J. A. (2006). *Elements of information theory*. John Wiley & Sons, 2 edition.

**Examples**

```
data(f125)
data(f125_enum)

plans_05 <- f125_enum$plans[, f125_enum$pop_dev <= 0.05]
distances <- redist.distances(plans_05)
distances$Hamming[1:5, 1:5]
```

---

plot.redist\_classified

*Plot a plan classification*

---

**Description**

Plot a plan classification

**Usage**

```
## S3 method for class 'redist_classified'
plot(x, plans, shp, type = "fill", which = NULL, ...)
```

**Arguments**

x	a redist_classified object, the output of <code>classify_plans()</code> .
plans	a <code>redist_plans</code> object.
shp	a shapefile or <code>redist_map</code> object.
type	either "line" or "fill". Passed on to <code>compare_plans()</code> as plot.
which	indices of the splits to plot. Defaults to all
...	passed on to <code>compare_plans()</code>

**Value**

ggplot comparison plot

---

plot.redist\_constr      *Visualize constraints*

---

## Description

Plots the constraint strength versus some running variable. Currently supports visualizing the `grp_hinge`, `grp_inv_hinge`, and `grp_pow` constraints.

## Usage

```
## S3 method for class 'redist_constr'  
plot(x, y, type = "group", xlim = c(0, 1), ...)
```

## Arguments

<code>x</code>	A <a href="#">redist_constr</a> object.
<code>y</code>	Ignored.
<code>type</code>	What type of constraint to visualize. Currently supports only "group", for visualizing constraint strength by group share.
<code>xlim</code>	Range of group shares to visualize.
<code>...</code>	additional arguments (ignored)

## Value

A ggplot object

## Examples

```
data(iowa)  
iowa_map <- redist_map(iowa, existing_plan = cd_2010, pop_tol = 0.05)  
constr <- redist_constr(iowa_map)  
constr <- add_constr_grp_hinge(constr, strength = 30,  
                             dem_08, tot_08, tgts_group = 0.5)  
constr <- add_constr_grp_hinge(constr, strength = -20,  
                             dem_08, tot_08, tgts_group = 0.3)  
plot(constr)
```

---

plot.redist_map	<i>Plot a redist_map</i>
-----------------	--------------------------

---

## Description

Plot a redist\_map

## Usage

```
## S3 method for class 'redist_map'  
plot(x, fill = NULL, by_distr = FALSE, adj = FALSE, ...)
```

## Arguments

x	the redist_map object
fill	<a href="#">&lt;data-masking&gt;</a> If present, will be used to color the map units. If using data masking, may need to explicitly name argument fill=... in non-interactive contexts to avoid S3 generic issues.
by_distr	if TRUE and fill is not missing and, color by district and indicate the fill variable by shading.
adj	if TRUE, force plotting the adjacency graph. Overrides by_distr.
...	passed on to <a href="#">redist.plot.map</a> (or <a href="#">redist.plot.adj</a> if adj=TRUE). Useful parameters may include zoom_to, boundaries, and title.

## Value

ggplot2 object

## Examples

```
data(f125)  
d <- redist_map(f125, ndists = 3, pop_tol = 0.05)  
plot(d)  
plot(d, BlackPop/pop)
```

```
data(f125_enum)  
f125$dist <- f125_enum$plans[, 5118]  
d <- redist_map(f125, existing_plan = dist)  
plot(d)
```

---

`plot.redist_plans`      *Summary plots for [\link{redist\\_plans}](#)*

---

### Description

If no arguments are passed, defaults to plotting the sampling weights for the [redist\\_plans](#) object. If no weights exist, plots district populations.

### Usage

```
## S3 method for class 'redist_plans'
plot(x, ..., type = "distr_qtys")
```

### Arguments

`x`                    the [redist\\_plans](#) object.  
`...`                passed on to the underlying function  
`type`                the name of the plotting function to use. Will have `redist.plot.`, prepended to it; e.g., use `type="plans"` to call [redist.plot.plans](#).

---

`prec_assignment`      *Extract the district assignments for a precinct across all simulated plans*

---

### Description

Extract the district assignments for a precinct across all simulated plans

### Usage

```
prec_assignment(prec, .data = pl())
```

### Arguments

`prec`                the precinct number  
`.data`               a [redist\\_plans](#) object

### Value

integer vector, a row from a plans matrix

---

```
prec_cooccurrence      Compute a matrix of precinct co-occurrences
```

---

**Description**

For a map with  $n$  precincts Returns an  $n$ -by- $n$  matrix, where each entry measures the fraction of the plans in which the row and column precincts were in the same district.

**Usage**

```
prec_cooccurrence(plans, which = NULL, sampled_only = TRUE, ncores = 1)
```

**Arguments**

`plans` a `redist_plans` object.  
`which` `<data-masking>` which plans to compute the co-occurrence over. Defaults to all.  
`sampled_only` if TRUE, do not include reference plans.  
`ncores` the number of parallel cores to use in the computation.

**Value**

a symmetric matrix the size of the number of precincts.

---

```
print.redist_classified
      Print redist_classified objects
```

---

**Description**

Print `redist_classified` objects

**Usage**

```
## S3 method for class 'redist_classified'
print(x, ...)
```

**Arguments**

`x` `redist_classified` object  
`...` additional arguments

**Value**

prints to console

print.redist\_constr     *Generic to print redist\_constr*

---

**Description**

Generic to print redist\_constr

**Usage**

```
## S3 method for class 'redist_constr'  
print(x, header = TRUE, details = TRUE, ...)
```

**Arguments**

x	redist_constr
header	if FALSE, then suppress introduction / header line
details	if FALSE, then suppress the details of each constraint
...	additional arguments

**Value**

Prints to console and returns input redist\_constr

---

print.redist\_map     *Generic to print redist\_map*

---

**Description**

Generic to print redist\_map

**Usage**

```
## S3 method for class 'redist_map'  
print(x, ...)
```

**Arguments**

x	redist_map
...	additional arguments

**Value**

Prints to console and returns input redist\_map

---

```
print.redist_plans      Print method for redist_plans
```

---

**Description**

Print method for redist\_plans

**Usage**

```
## S3 method for class 'redist_plans'
print(x, ...)
```

**Arguments**

```
x                a redist\_plans object
...              additional arguments (ignored)
```

**Value**

The original object, invisibly.

---

```
proj              Calculate Projective Distributions, Averages, and Contrasts for a
                  Summary Statistic
```

---

**Description**

The *projective distribution* of a district-level summary statistic (McCartan 2024) is the distribution of values of that statistic across a set of plans for the district each precinct belongs to. The *projective average* of a statistic is the average value of the projective distribution in each precinct. A *projective contrast* is the difference between the projective average for a single plan and the projective average for an ensemble of sampled plans.

It is very important to properly account for variation in the projective distribution when looking at projective contrasts. The `pfdr` argument to `proj_contr()` will calculate q-values for each precinct that can be used to control the positive false discovery rate (pFDR) to avoid being misled by this variation. See `redist.plot.contr_pfdr()` for a way to automatically plot projective contrasts with this false discovery rate control.

**Usage**

```
proj_distr(plans, x, draws = NA)

proj_avg(plans, x, draws = NA)

proj_contr(plans, x, compare = NA, draws = NA, norm = FALSE, pfdr = FALSE)
```

**Arguments**

plans	A <a href="#">redist_plans</a> object.
x	A district-level summary statistic calculated from the plans object. Tidy-evaluated in plans.
draws	which draws/samples to include in the projective distribution. NULL will include all draws, including reference plans. The special value NA will include all sampled (non-reference) draws. An integer, logical, or character vector indicating specific draws may also be provided.
compare	The plan to compare to the rest of the ensemble (which is controlled by draws). Defaults to the first reference plan, if any exists
norm	If TRUE, normalize the contrast by the standard deviation of the projective distribution, precinct-wise. This will make the projective contrast in terms of z-scores.
pfdr	If TRUE, calculate q-values for each precinct that can be used to control the positive false discovery rate (pFDR) at a given level by thresholding the q-values at that level. Q-values are stored as the "q" attribute on the returned vector. Requires the <code>matrixStats</code> package be installed.

**Value**

proj\_distr: A matrix with a row for each precinct (row in the map object) and a column for every draw described by draws.

proj\_avg: A numeric vector of length matching the number of precincts.

proj\_contr: A numeric vector of length matching the number of precincts, optionally with a "q" attribute containing q-values.

**References**

McCartan, C. (2024). Projective Averages for Summarizing Redistricting Ensembles. *arXiv preprint*. Available at <https://arxiv.org/pdf/2401.06381>.

**Examples**

```
data(iowa)
map <- redist_map(iowa, existing_plan = cd_2010, pop_tol = 0.01)
plans <- redist_smc(map, 50, silent = TRUE)
plans$dem <- group_frac(map, dem_08, tot_08, plans)

proj_distr(plans, dem)[,1] # a 99-by-50 matrix, just showing first column
plot(map, proj_avg(plans, dem))
plot(map, proj_contr(plans, dem))
plot(map, proj_contr(plans, dem, compare="cd_2010"))
```

---

pullback	<i>Pull back plans to unmerged units</i>
----------	--

---

**Description**

Merging map units through `merge_by` or `dplyr::summarize` changes the indexing of each unit. Use this function to take a set of redistricting plans from a `redist` algorithm and re-index them to be compatible with the original set of units.

**Usage**

```
pullback(plans, map = NULL)
```

**Arguments**

<code>plans</code>	a <code>redist_plans</code> object
<code>map</code>	optionally, a <code>redist_map</code> object, which will be used to set the new population vector

**Value**

a new, re-indexed, `redist_plans` object

---

<code>rbind.redist_plans</code>	<i>Combine multiple sets of redistricting plans</i>
---------------------------------	---

---

**Description**

Only works when all the sets are compatible—generated from the same map, with the same number of districts. Sets of plans will be indexed by the chain column.

**Usage**

```
## S3 method for class 'redist_plans'
rbind(..., deparse.level = 1)
```

**Arguments**

<code>...</code>	The <code>redist_plans</code> objects to combine. If named arguments are provided, the names will be used in the chain column; otherwise, numbers will be used for the chain column.
<code>deparse.level</code>	Ignored.

**Value**

A new `redist_plans` object.

---

redist.adjacency      *Adjacency List functionality for redist*

---

**Description**

Creates an adjacency list that is zero indexed with no skips

**Usage**

```
redist.adjacency(shp, plan)
```

**Arguments**

shp                    A SpatialPolygonsDataFrame or sf object. Required.  
 plan                   A numeric vector (if only one map) or matrix with one row

**Value**

Adjacency list

---

redist.calc.frontier.size  
*Calculate Frontier Size*

---

**Description**

Calculate Frontier Size

**Usage**

```
redist.calc.frontier.size(ordered_path)
```

**Arguments**

ordered\_path      path to ordered path created by redist.prep.enumpart

**Value**

List, four objects

- max numeric, maximum frontier size
- average numeric, average frontier size
- average\_sq numeric, average((frontier size)^2)
- sequence numeric vector, lists out all sizes for every frontier

**Examples**

```
## Not run:
data(f125)
adj <- redist.adjacency(f125)
redist.prep.enumpart(adj, "unordered", "ordered")
redist.calc.frontier.size("ordered")

## End(Not run)
```

---

```
redist.coarsen.adjacency
      Coarsen Adjacency List
```

---

**Description**

Coarsen Adjacency List

**Usage**

```
redist.coarsen.adjacency(adj, groups)
```

**Arguments**

adj	A zero-indexed adjacency list. Required.
groups	integer vector of elements of adjacency to group

**Value**

adjacency list coarsened

---

```
redist.combine.mpi      Combine successive runs of redist.mcmc.mpi
```

---

**Description**

redist.combine.mpi is used to combine successive runs of redist.mcmc.mpi into a single data object

**Usage**

```
redist.combine.mpi(savename, nloop, nthin, tempadj)
```

**Arguments**

<code>savename</code>	The name (without the loop or <code>.RData</code> suffix) of the saved simulations.
<code>nloop</code>	The number of loops being combined.
<code>nthin</code>	How much to thin the simulations being combined.
<code>tempadj</code>	The temperature adjacency object saved by <code>redist.mcmc.mpi</code> .

**Details**

This function allows users to combine multiple successive runs of `redist.mcmc.mpi` into a single `redist` object for analysis.

**Value**

`redist.combine.mpi` returns an object of class "redist". The object `redist` is a list that contains the following components (the inclusion of some components is dependent on whether tempering techniques are used):

<code>plans</code>	Matrix of congressional district assignments generated by the algorithm. Each row corresponds to a geographic unit, and each column corresponds to a simulation.
<code>distance_parity</code>	Vector containing the maximum distance from parity for a particular simulated redistricting plan.
<code>mhdecisions</code>	A vector specifying whether a proposed redistricting plan was accepted (1) or rejected (0) in a given iteration.
<code>mhprob</code>	A vector containing the Metropolis-Hastings acceptance probability for each iteration of the algorithm.
<code>pparam</code>	A vector containing the draw of the $p$ parameter for each simulation, which dictates the number of swaps attempted.
<code>constraint_pop</code>	A vector containing the value of the population constraint for each accepted redistricting plan.
<code>constraint_compact</code>	A vector containing the value of the compactness constraint for each accepted redistricting plan.
<code>constraint_vra</code>	A vector containing the value of the vra constraint for each accepted redistricting plan.
<code>constraint_similar</code>	A vector containing the value of the similarity constraint for each accepted redistricting plan.
<code>constraint_qps</code>	A vector containing the value of the QPS constraint for each accepted redistricting plan.
<code>beta_sequence</code>	A vector containing the value of beta for each iteration of the algorithm. Returned when tempering is being used.
<code>mhdecisions_beta</code>	A vector specifying whether a proposed beta value was accepted (1) or rejected (0) in a given iteration of the algorithm. Returned when tempering is being used.

mhprob\_beta      A vector containing the Metropolis-Hastings acceptance probability for each iteration of the algorithm. Returned when tempering is being used.

## References

Fifield, Benjamin, Michael Higgins, Kosuke Imai and Alexander Tarr. (2016) "A New Automated Redistricting Simulator Using Markov Chain Monte Carlo." Working Paper. Available at <http://imai.princeton.edu/research/files/redist.pdf>.

## Examples

```
## Not run:
# Cannot run on machines without Rmpi
data(fl25)
data(fl25_enum)
data(fl25_adj)

## Code to run the simulations in Figure 4 in Fifield, Higgins, Imai and
## Tarr (2015)

## Get an initial partition
init_plan <- fl25_enum$plans[, 5118]

## Run the algorithm
redist.mcmc.mpi(adj = fl25_adj, total_pop = fl25$pop,
  init_plan = init_plan, nsims = 10000, nloops = 2, savename = "test")
out <- redist.combine.mpi(savename = "test", nloop = 2,
  nthin = 10, tempadj = tempAdjMat)

## End(Not run)
```

---

redist.constraint.helper

*Create Constraints for SMC*

---

## Description

Create Constraints for SMC

## Usage

```
redist.constraint.helper(
  constraints = "vra",
  tgt_min = 0.55,
  group_pop,
  total_pop,
  ndists,
  nmmd,
  strength_vra = 2500,
```

```
    pow_vra = 1.5
  )
```

### Arguments

constraints	Vector of constraints to include. Currently only 'vra' implemented.
tgt_min	Defaults to 0.55. If 'vra' included, the minority percent to encourage in each district.
group_pop	A vector of populations for some subgroup of interest.
total_pop	A vector containing the populations of each geographic unit.
ndists	The total number of districts.
nmmd	The number of majority minority districts to target for 'vra' constraint
strength_vra	The strength of the 'vra' constraint. Defaults to 2500.
pow_vra	The exponent for the 'vra' constraint. Defaults to 1.5.

### Value

list of lists for each constraint selected

---

redist.county.id	<i>Create County IDs</i>
------------------	--------------------------

---

### Description

Create County IDs

### Usage

```
redist.county.id(counties)
```

### Arguments

counties	vector of counties, required.
----------	-------------------------------

### Value

A vector with an ID that corresponds from 1:n counties

### Examples

```
set.seed(2)
counties <- sample(c(rep("a", 20), rep("b", 5)))
redist.county.id(counties)
```

---

redist.county.relabel *Relabel Discontinuous Counties*

---

### Description

Relabel Discontinuous Counties

### Usage

```
redist.county.relabel(adj, counties, simplify = TRUE)
```

### Arguments

adj	adjacency list
counties	character vector of county names
simplify	boolean - TRUE returns a numeric vector of ids, while FALSE appends a number when there are multiple connected components.

### Value

character vector of county names

### Examples

```
set.seed(2)
data(f125)
data(f125_adj)
counties <- sample(c(rep("a", 20), rep("b", 5)))
redist.county.relabel(f125_adj, counties)
```

---

redist.crsg *Redistricting via Compact Random Seed and Grow Algorithm*

---

### Description

redist.crsg generates redistricting plans using a random seed a grow algorithm. This is the compact districting algorithm described in Chen and Rodden (2013).

**Usage**

```
redist.crsg(
  adj,
  total_pop,
  shp,
  ndists,
  pop_tol,
  verbose = TRUE,
  maxiter = 5000
)
```

**Arguments**

adj	List of length N, where N is the number of precincts. Each list element is an integer vector indicating which precincts that precinct is adjacent to. It is assumed that precinct numbers start at 0.
total_pop	numeric vector of length N, where N is the number of precincts. Each element lists the population total of the corresponding precinct, and is used to enforce pop_tol constraints.
shp	An sf dataframe to compute area and centroids with.
ndists	integer, the number of districts we want to partition the precincts into.
pop_tol	numeric, indicating how close district population targets have to be to the target population before algorithm converges. pop_tol=0.05 for example means that all districts must be between 0.95 and 1.05 times the size of target.pop in population size.
verbose	boolean, indicating whether the time to run the algorithm is printed.
maxiter	integer, indicating maximum number of iterations to attempt before convergence to population constraint fails. If it fails once, it will use a different set of start values and try again. If it fails again, redist.rsg() returns an object of all NAs, indicating that use of more iterations may be advised. Default is 5000.

**Value**

list, containing three objects containing the completed redistricting plan.

- plan: A vector of length N, indicating the district membership of each precinct.
- district\_list A list of length Ndistrict. Each list contains a vector of the precincts in the respective district.
- district\_pop A vector of length Ndistrict, containing the population totals of the respective districts.

**References**

Jowei Chen and Jonathan Rodden (2013) "Unintentional Gerrymandering: Political Geography and Electoral Bias in Legislatures." *Quarterly Journal of Political Science*. 8(3): 239-269.

## Examples

```
data("f125")
adj <- redist.adjacency(f125)
redist.crsq(adj = adj, total_pop = f125$pop, shp = f125, ndists = 2, pop_tol = .1)
```

---

redist.diagplot      *Diagnostic plotting functionality for MCMC redistricting.*

---

## Description

redist.diagplot generates several common MCMC diagnostic plots.

## Usage

```
redist.diagplot(sumstat,
plot = c("trace", "autocorr", "densplot", "mean", "gelmanrubin"),
logit = FALSE, savename = NULL)
```

## Arguments

sumstat	A vector, list, mcmc or mcmc.list object containing a summary statistic of choice.
plot	The type of diagnostic plot to generate: one of "trace", "autocorr", "densplot", "mean", "gelmanrubin". If plot = "gelmanrubin", the input sumstat must be of class mcmc.list or list.
logit	Flag for whether to apply the logistic transformation for the summary statistic. The default is FALSE.
savename	Filename to save the plot. Default is NULL.

## Details

This function allows users to generate several standard diagnostic plots from the MCMC literature, as implemented by Plummer et. al (2006). Diagnostic plots implemented include trace plots, autocorrelation plots, density plots, running means, and Gelman-Rubin convergence diagnostics (Gelman & Rubin 1992).

## Value

Returns a plot of file type .pdf.

**References**

Fifield, Benjamin, Michael Higgins, Kosuke Imai and Alexander Tarr. (2016) "A New Automated Redistricting Simulator Using Markov Chain Monte Carlo." Working Paper. Available at <http://imai.princeton.edu/research/files/redist.pdf>.

Gelman, Andrew and Donald Rubin. (1992) "Inference from iterative simulations using multiple sequences (with discussion)." *Statistical Science*.

Plummer, Martin, Nicky Best, Kate Cowles and Karen Vines. (2006) "CODA: Convergence Diagnosis and Output Analysis for MCMC." *R News*.

**Examples**

```

data(f125)
data(f125_enum)
data(f125_adj)

## Get an initial partition
init_plan <- f125_enum$plans[, 5118]
f125$init_plan <- init_plan

## 25 precinct, three districts - no pop constraint ##
fl_map <- redist_map(f125, existing_plan = 'init_plan', adj = f125_adj)
alg_253 <- redist_flip(fl_map, nsims = 10000)

## Get Republican Dissimilarity Index from simulations
rep_dmi_253 <- redistmetrics::seg_dissim(alg_253, f125, mccain, pop) |>
  redistmetrics::by_plan(ndists = 3)

## Generate diagnostic plots
redist.diagplot(rep_dmi_253, plot = "trace")
redist.diagplot(rep_dmi_253, plot = "autocorr")
redist.diagplot(rep_dmi_253, plot = "densplot")
redist.diagplot(rep_dmi_253, plot = "mean")

## Gelman Rubin needs two chains, so we run a second
alg_253_2 <- redist_flip(fl_map, nsims = 10000)

rep_dmi_253_2 <- redistmetrics::seg_dissim(alg_253_2, f125, mccain, pop) |>
  redistmetrics::by_plan(ndists = 3)

## Make a list out of the objects:
rep_dmi_253_list <- list(rep_dmi_253, rep_dmi_253_2)

## Generate Gelman Rubin diagnostic plot
redist.diagplot(sumstat = rep_dmi_253_list, plot = "gelmanrubin")

```

---

 redist.dist.pop.overlap

*Compare the Population Overlap Across Plans at the District Level*


---

### Description

This implements Crespin's 2005 measure of district continuity, as applied to the geographies represented by a plan, typically precincts or voting districts. This implementation assumes none of the precincts in `plan_old` or `plan_new` are split.

### Usage

```
redist.dist.pop.overlap(plan_old, plan_new, total_pop, normalize_rows = TRUE)
```

### Arguments

<code>plan_old</code>	The reference or original plan to compare against
<code>plan_new</code>	The new plan to compare to the reference plan
<code>total_pop</code>	The total population by precinct This can also take a <code>redist_map</code> object and will use the population in that object. If nothing is provided, it weights all entries in <code>plan</code> equally.
<code>normalize_rows</code>	Default TRUE. Normalize populations by row. If FALSE, normalizes by column. If NULL, does not normalize.

### Value

matrix with `length(unique(plan_old))` rows and `length(unique(plan_new))` columns

### References

"Using Geographic Information Systems to Measure District Change, 2000-02", Michael Crespin, *Political Analysis* (2005) 13(3): 253-260

### Examples

```
set.seed(5)
data(iowa)
iowa_map <- redist_map(iowa, total_pop = pop, pop_tol = 0.01, ndists = 4)
plans <- redist_smc(iowa_map, 2)
plans_mat <- get_plans_matrix(plans)
ov <- redist.dist.pop.overlap(plans_mat[, 1], plans_mat[, 2], iowa_map)
round(ov, 2)

ov_col <- redist.dist.pop.overlap(plans_mat[, 1], plans_mat[, 2], iowa_map, normalize_rows = FALSE)
round(ov_col, 2)

ov_un_norm <- redist.dist.pop.overlap(plans_mat[, 1], plans_mat[, 2],
  iowa_map, normalize_rows = NULL)
```

```

round(ov_un_norm, 2)

iowa_map_5 <- iowa_map <- redist_map(iowa, total_pop = pop, pop_tol = 0.01, ndists = 5)
plan_5 <- get_plans_matrix(redist_smc(iowa_map_5, 1))
ov4_5 <- redist.dist.pop.overlap(plans_mat[, 1], plan_5, iowa_map)
round(ov4_5, 2)

```

---

`redist.district.splits`

*Counts the Number of Counties within a District*

---

### Description

Counts the total number of counties that are found within a district. This does not subtract out the number of counties that are found completely within a district.

### Usage

```
redist.district.splits(plans, counties)
```

### Arguments

<code>plans</code>	A numeric vector (if only one map) or matrix with one row for each precinct and one column for each map. Required.
<code>counties</code>	A vector of county names or county ids.

### Value

integer matrix where each district is a

### Examples

```

data(iowa)
ia <- redist_map(iowa, existing_plan = cd_2010, total_pop = pop, pop_tol = 0.01)
plans <- redist_smc(ia, 50, silent = TRUE)
#old redist.district.splits(plans, ia$region)
splits_count(plans, ia, region)

```

---

redist.enumpart      *Enumerate All Partitions (Fifield et al. 2020)*

---

### Description

Single function for standard enumeration analysis, using ZDD methodology (Fifield, Imai, Kawahara, and Kenny 2020).

### Usage

```
redist.enumpart(
  adj,
  unordered_path,
  ordered_path,
  out_path,
  ndists = 2,
  all = TRUE,
  n = NULL,
  weight_path = NULL,
  lower = NULL,
  upper = NULL,
  init = FALSE,
  read = TRUE,
  total_pop = NULL
)
```

### Arguments

adj	zero indexed adjacency list.
unordered_path	valid path to output the unordered adjacency map to
ordered_path	valid path to output the ordered adjacency map to
out_path	Valid path to output the enumerated districts
ndists	number of districts to enumerate
all	boolean. TRUE outputs all districts. FALSE samples n districts.
n	integer. Number of districts to output if all is FALSE. Returns districts selected from uniform random distribution.
weight_path	A path (not including ".dat") to a space-delimited file containing a vector of vertex weights, to be used along with lower and upper.
lower	A lower bound on each partition's total weight, implemented by rejection sampling.
upper	An upper bound on each partition's total weight.
init	Runs redist.init.enumpart. Defaults to false. Should be run on first use.
read	boolean. Defaults to TRUE. reads
total_pop	the vector of precinct populations

**Value**

List with entries district\_membership and parity.

**References**

Fifield, B., Imai, K., Kawahara, J., & Kenny, C. T. (2020). The essential role of empirical validation in legislative redistricting simulation. *Statistics and Public Policy*, 7(1), 52-68.

---

redist.find.target	<i>Find Majority Minority Remainder</i>
--------------------	---

---

**Description**

Given a percent goal for majority minority districts, this computes the average value of minority in non-majority minority districts. This value is "tgt\_other" in redist\_flip and redist\_smc.

**Usage**

```
redist.find.target(tgt_min, group_pop, total_pop, ndists, nmmd)
```

**Arguments**

tgt_min	target group population for majority minority district
group_pop	A vector of populations for some subgroup of interest.
total_pop	A vector containing the populations of each geographic unit.
ndists	The number of congressional districts.
nmmd	The number of majority minority districts.

**Value**

numeric value to target

---

redist.findparams	<i>Run parameter testing for redist_flip</i>
-------------------	--

---

**Description**

redist.findparams is used to find optimal parameter values of redist\_flip for a given map.

**Usage**

```

redist.findparams(
  map,
  nsims,
  init_plan = NULL,
  adapt_lambda = FALSE,
  adapt_eprob = FALSE,
  params,
  ssdmat = NULL,
  group_pop = NULL,
  counties = NULL,
  nstartval_store = 1,
  maxdist_startval = 100,
  maxiterrsg = 5000,
  report_all = TRUE,
  parallel = FALSE,
  ncores = NULL,
  log = FALSE,
  verbose = TRUE
)

```

**Arguments**

map	A <a href="#">redist_map</a> object.
nsims	The number of simulations run before a save point.
init_plan	A vector containing the congressional district labels of each geographic unit. The default is NULL. If not provided, random and contiguous congressional district assignments will be generated using <code>redist.rsg</code> .
adapt_lambda	Whether to adaptively tune the lambda parameter so that the Metropolis-Hastings acceptance probability falls between 20% and 40%. Default is FALSE.
adapt_eprob	Whether to adaptively tune the edgcut probability parameter so that the Metropolis-Hastings acceptance probability falls between 20% and 40%. Default is FALSE.
params	A matrix of parameter values to test, such as the output of <code>expand.grid</code> . Parameters accepted for <code>params</code> include <code>eprob</code> , <code>lambda</code> , <code>pop_tol</code> , <code>beta</code> , and <code>constraint</code> .
ssdmat	A matrix of squared distances between geographic units. The default is NULL.
group_pop	A vector of populations for some sub-group of interest. The default is NULL.
counties	A vector of county membership assignments. The default is NULL.
nstartval_store	The number of maps to sample from the preprocessing chain for use as starting values in future simulations. Default is 1.
maxdist_startval	The maximum distance from the starting map that sampled maps should be. Default is 100 (no restriction).
maxiterrsg	Maximum number of iterations for random seed-and-grow algorithm to generate starting values. Default is 5000.

<code>report_all</code>	Whether to report all summary statistics for each set of parameter values. Default is TRUE.
<code>parallel</code>	Whether to run separate parameter settings in parallel. Default is FALSE.
<code>ncores</code>	Number of parallel tasks to run, declared outside of the function. Default is NULL.
<code>log</code>	Whether to open a log to track progress for each parameter combination being tested. Default is FALSE.
<code>verbose</code>	Whether to print additional information about the tests. Default is TRUE.

### Details

This function allows users to test multiple parameter settings of `redist_flip` in preparation for a longer run for analysis.

### Value

`redist.findparams` returns a print-out of summary statistics about each parameter setting.

### References

Fifield, Benjamin, Michael Higgins, Kosuke Imai and Alexander Tarr. (2016) "A New Automated Redistricting Simulator Using Markov Chain Monte Carlo." Working Paper. Available at <http://imai.princeton.edu/research/files/redist.pdf>.

### Examples

```
data(fl25)
data(fl25_enum)
data(fl25_adj)

## Get an initial partition
init_plan <- fl25_enum$plans[, 5118]

params <- expand.grid(eprob = c(.01, .05, .1))

# Make map
map_fl <- redist_map(fl25, ndists = 3, pop_tol = 0.2)
## Run the algorithm
redist.findparams(map_fl,
  init_plan = init_plan, nsims = 10000, params = params)
```

---

```
redist.init.enumpart Initialize enumpart
```

---

**Description**

This ensures that the enumerate partitions programs is prepared to run. This must be run once per install of the redist package.

**Usage**

```
redist.init.enumpart()
```

**Value**

0 on success

**References**

Benjamin Fifield, Kosuke Imai, Jun Kawahara, and Christopher T Kenny. "The Essential Role of Empirical Validation in Legislative Redistricting Simulation." Forthcoming, *Statistics and Public Policy*.

**Examples**

```
## Not run:
redist.init.enumpart()

## End(Not run)
```

---

```
redist.ipw Inverse probability reweighting for MCMC Redistricting
```

---

**Description**

redist.ipw properly weights and resamples simulated redistricting plans so that the set of simulated plans resemble a random sample from the underlying distribution. redist.ipw is used to correct the sample when population parity, geographic compactness, or other constraints are implemented.

**Usage**

```
redist.ipw(
  plans,
  resampleconstraint = c("pop_dev", "edges_removed", "segregation", "status_quo"),
  targetbeta,
  targetpop = NULL,
  temper = 0
)
```

**Arguments**

plans	An object of class <code>redist_plans</code> from <code>redist_flip()</code> .
resampleconstraint	The constraint implemented in the simulations: one of "pop", "compact", "segregation", or "similar".
targetbeta	The target value of the constraint.
targetpop	The desired level of population parity. <code>targetpop = 0.01</code> means that the desired distance from population parity is 1%. The default is <code>NULL</code> .
temper	A flag for whether simulated tempering was used to improve the mixing of the Markov Chain. The default is 1.

**Details**

This function allows users to resample redistricting plans using inverse probability weighting techniques described in Rubin (1987). This techniques reweights and resamples redistricting plans so that the resulting sample is representative of a random sample from the uniform distribution.

**Value**

`redist.ipw` returns an object of class "redist". The object `redist` is a list that contains the following components (the inclusion of some components is dependent on whether tempering techniques are used):

plans	Matrix of congressional district assignments generated by the algorithm. Each row corresponds to a geographic unit, and each column corresponds to a simulation.
distance_parity	Vector containing the maximum distance from parity for a particular simulated redistricting plan.
mhdecisions	A vector specifying whether a proposed redistricting plan was accepted (1) or rejected (0) in a given iteration.
mhprob	A vector containing the Metropolis-Hastings acceptance probability for each iteration of the algorithm.
pparam	A vector containing the draw of the $p$ parameter for each simulation, which dictates the number of swaps attempted.
constraint_pop	A vector containing the value of the population constraint for each accepted redistricting plan.
constraint_compact	A vector containing the value of the compactness constraint for each accepted redistricting plan.
constraint_segregation	A vector containing the value of the segregation constraint for each accepted redistricting plan.
constraint_similar	A vector containing the value of the similarity constraint for each accepted redistricting plan.

constraint_vra	A vector containing the value of the vra constraint for each accepted redistricting plan.
constraint_partisan	A vector containing the value of the partisan constraint for each accepted redistricting plan.
constraint_minority	A vector containing the value of the minority constraint for each accepted redistricting plan.
constraint_hinge	A vector containing the value of the hinge constraint for each accepted redistricting plan.
constraint_qps	A vector containing the value of the QPS constraint for each accepted redistricting plan.
beta_sequence	A vector containing the value of beta for each iteration of the algorithm. Returned when tempering is being used.
mhdecisions_beta	A vector specifying whether a proposed beta value was accepted (1) or rejected (0) in a given iteration of the algorithm. Returned when tempering is being used.
mhprob_beta	A vector containing the Metropolis-Hastings acceptance probability for each iteration of the algorithm. Returned when tempering is being used.

## References

Fifield, Benjamin, Michael Higgins, Kosuke Imai and Alexander Tarr. (2016) "A New Automated Redistricting Simulator Using Markov Chain Monte Carlo." Working Paper. Available at <http://imai.princeton.edu/research/files/redist.pdf>.

Rubin, Donald. (1987) "Comment: A Noniterative Sampling/Importance Resampling Alternative to the Data Augmentation Algorithm for Creating a Few Imputations when Fractions of Missing Information are Modest: the SIR Algorithm." Journal of the American Statistical Association.

## Examples

```
data(iowa)
map_ia <- redist_map(iowa, existing_plan = cd_2010, pop_tol = 0.01)
cons <- redist_constr(map_ia)
cons <- add_constr_pop_dev(cons, strength = 5.4)
alg <- redist_flip(map_ia, nsims = 500, constraints = cons)

alg_ipw <- redist.ipw(plans = alg,
  resampleconstraint = "pop_dev",
  targetbeta = 1,
  targetpop = 0.05)
```

---

redist.mcmc.mpi      *MCMC Redistricting Simulator using MPI*

---

### Description

redist.mcmc.mpi is used to simulate Congressional redistricting plans using Markov Chain Monte Carlo methods.

### Usage

```
redist.mcmc.mpi(
  adj,
  total_pop,
  nsims,
  ndists = NA,
  init_plan = NULL,
  loopscompleted = 0,
  nloop = 1,
  nthin = 1,
  eprob = 0.05,
  lambda = 0,
  pop_tol = NA,
  group_pop = NA,
  areasvec = NA,
  counties = NA,
  borderlength_mat = NA,
  ssdmat = NA,
  compactness_metric = "fryer-holden",
  rngseed = NA,
  constraint = NA,
  constraintweights = NA,
  betaseq = "powerlaw",
  betaseqlength = 10,
  adjswaps = TRUE,
  freq = 100,
  savename = NA,
  maxiterrsg = 5000,
  verbose = FALSE,
  cities = NULL
)
```

### Arguments

adj	An adjacency matrix, list, or object of class "SpatialPolygonsDataFrame."
total_pop	A vector containing the populations of each geographic unit.
nsims	The number of simulations run before a save point.

ndists	The number of congressional districts. The default is NULL.
init_plan	A vector containing the congressional district labels of each geographic unit. The default is NULL. If not provided, random and contiguous congressional district assignments will be generated using <code>redist.rsg</code> .
loopscompleted	Number of save points reached by the algorithm. The default is 0.
nloop	The total number of save points for the algorithm. The default is 1. Note that the total number of simulations run will be $nsims * nloop$ .
nthin	The amount by which to thin the Markov Chain. The default is 1.
eprob	The probability of keeping an edge connected. The default is 0.05.
lambda	The parameter determining the number of swaps to attempt each iteration of the algorithm. The number of swaps each iteration is equal to $Pois(\lambda) + 1$ . The default is 0.
pop_tol	The strength of the hard population constraint. $pop\_tol = 0.05$ means that any proposed swap that brings a district more than 5% rejected. The default is NULL.
group_pop	A vector of populations for some sub-group of interest. The default is NULL.
areasvec	A vector of precinct areas for discrete Polsby-Popper. The default is NULL.
counties	A vector of county membership assignments. The default is NULL.
borderlength_mat	A matrix of border length distances, where the first two columns are the indices of precincts sharing a border and the third column is its distance. Default is NULL.
ssdmat	A matrix of squared distances between geographic units. The default is NULL.
compactness_metric	The compactness metric to use when constraining on compactness. Default is <code>fryer-holden</code> , the other implemented option is <code>polsby-popper</code> .
rngseed	Allows the user to set the seed for the simulations. Default is NULL.
constraint	Which constraint to apply. Accepts any combination of <code>compact</code> , <code>vra</code> , <code>population</code> , <code>similarity</code> , or <code>none</code> (no constraint applied). The default is NULL.
constraintweights	The weights to apply to each constraint. Should be a vector the same length as constraint. Default is NULL.
betaseq	Sequence of beta values for tempering. The default is <code>powerlaw</code> (see Fifield et al (2015) for details).
betaseqlength	Length of beta sequence desired for tempering. The default is 10.
adjswaps	Flag to restrict swaps of beta so that only values adjacent to current constraint are proposed. The default is TRUE.
freq	Frequency of between-chain swaps. Default to once every 100 iterations
savename	Filename to save simulations. Default is NULL.
maxiterrsg	Maximum number of iterations for random seed-and-grow algorithm to generate starting values. Default is 5000.
verbose	Whether to print initialization statement. Default is TRUE.
cities	integer vector of cities for QPS constraint.

**Details**

This function allows users to simulate redistricting plans using Markov Chain Monte Carlo methods. Several constraints corresponding to substantive requirements in the redistricting process are implemented, including population parity and geographic compactness. In addition, the function includes multiple-swap and parallel tempering functionality in MPI to improve the mixing of the Markov Chain.

**Value**

`redist.mcmc.mpi` returns an object of class "redist". The object `redist` is a list that contains the following components (the inclusion of some components is dependent on whether tempering techniques are used):

<code>partitions</code>	Matrix of congressional district assignments generated by the algorithm. Each row corresponds to a geographic unit, and each column corresponds to a simulation.
<code>distance_parity</code>	Vector containing the maximum distance from parity for a particular simulated redistricting plan.
<code>mhdecisions</code>	A vector specifying whether a proposed redistricting plan was accepted (1) or rejected (0) in a given iteration.
<code>mhprob</code>	A vector containing the Metropolis-Hastings acceptance probability for each iteration of the algorithm.
<code>pparam</code>	A vector containing the draw of the $p$ parameter for each simulation, which dictates the number of swaps attempted.
<code>constraint_pop</code>	A vector containing the value of the population constraint for each accepted redistricting plan.
<code>constraint_compact</code>	A vector containing the value of the compactness constraint for each accepted redistricting plan.
<code>constraint_vra</code>	A vector containing the value of the vra constraint for each accepted redistricting plan.
<code>constraint_similar</code>	A vector containing the value of the similarity constraint for each accepted redistricting plan.
<code>beta_sequence</code>	A vector containing the value of beta for each iteration of the algorithm. Returned when tempering is being used.
<code>mhdecisions_beta</code>	A vector specifying whether a proposed beta value was accepted (1) or rejected (0) in a given iteration of the algorithm. Returned when tempering is being used.
<code>mhprob_beta</code>	A vector containing the Metropolis-Hastings acceptance probability for each iteration of the algorithm. Returned when tempering is being used.

**References**

Fifield, Benjamin, Michael Higgins, Kosuke Imai and Alexander Tarr. (2016) "A New Automated Redistricting Simulator Using Markov Chain Monte Carlo." Working Paper. Available at <http://imai.princeton.edu/research/files/redist.pdf>.

**Examples**

```
## Not run:
# Cannot run on machines without Rmpi
data(fl25)
data(fl25_enum)
data(fl25_adj)

## Code to run the simulations in Figure 4 in Fifield, Higgins, Imai and
## Tarr (2015)

## Get an initial partition
init_plan <- fl25_enum$plans[, 5118]

## Run the algorithm
redist.mcmc.mpi(adj = fl25_adj, total_pop = fl25$pop,
  init_plan = init_plan, nsims = 10000, savename = "test")

## End(Not run)
```

---

redist.multisplits      *Counts the Number of Counties Split Between 3 or More Districts*

---

**Description**

Counts the total number of counties that are split across more than 2 districts.

**Usage**

```
redist.multisplits(plans, counties)
```

**Arguments**

plans	A numeric vector (if only one map) or matrix with one row for each precinct and one column for each map. Required.
counties	A vector of county names or county ids.

**Value**

integer matrix where each district is a

**Examples**

```
data(iowa)
ia <- redist_map(iowa, existing_plan = cd_2010, total_pop = pop, pop_tol = 0.01)
plans <- redist_smc(ia, 50, silent = TRUE)
#old redist.multisplits(plans, ia$region)
splits_multi(plans, ia, region)
```

---

redist.parity	<i>Calculates Maximum Deviation from Population Parity</i>
---------------	--

---

### Description

Computes the deviation from population parity from a plan. Higher values indicate that (at least) a single district in the map deviates from population parity. See Details.

### Usage

```
redist.parity(plans, total_pop)
```

```
plan_parity(map, .data = pl(), ...)
```

### Arguments

plans	A matrix with one row for each precinct and one column for each map. Required.
total_pop	A numeric vector with the population for every precinct.
map	a <a href="#">redist_map</a> object
.data	a <a href="#">redist_plans</a> object
...	passed on to redist.parity

### Details

With a map with pop representing the populations of each district, the deviation from population parity is given as  $\max(\text{abs}(\text{pop} - \text{parity}) / \text{parity})$  where  $\text{parity} = \text{sum}(\text{pop}) / \text{length}(\text{pop})$  is the population size for the average district. Therefore, the metric can be thought of as the maximum percent deviation from equal population. For example, a value of 0.03 in this metric indicates that all districts are within 3 percent of population parity.

### Value

numeric vector with the population parity for each column

---

redist.plot.adj	<i>Creates a Graph Overlay</i>
-----------------	--------------------------------

---

### Description

Creates a Graph Overlay

**Usage**

```
redist.plot.adj(
  shp,
  adj = NULL,
  plan = NULL,
  centroids = TRUE,
  drop = FALSE,
  plot_shp = TRUE,
  zoom_to = NULL,
  title = ""
)
```

**Arguments**

shp	A SpatialPolygonsDataFrame or sf object. Required.
adj	A zero-indexed adjacency list. Created with redist.adjacency if not supplied. Default is NULL.
plan	A numeric vector with one entry for each precinct in shp. Used to remove edges that cross boundaries. Default is NULL. Optional.
centroids	A logical indicating if centroids should be plotted. Default is TRUE.
drop	A logical indicating if edges that cross districts should be dropped. Default is FALSE.
plot_shp	A logical indicating if the shp should be plotted under the graph. Default is TRUE.
zoom_to	<a href="#">&lt;data-masking&gt;</a> An indexing vector of units to zoom the map to.
title	A string title of plot. Defaults to empty string. Optional.

**Value**

ggplot map

**Examples**

```
data(iowa)
redist.plot.adj(shp = iowa, plan = iowa$cd_2010)
```

---

```
redist.plot.contr_pfd
```

*Plot a Projective Contrast with positive False Discovery Rate (pFDR) Control*

---

**Description**

Plot a projective contrast on a map with areas selected by the pFDR control procedure hatched.

**Usage**

```
redist.plot.contr_pfd(
  map,
  contr,
  level = 0.05,
  density = 0.2,
  spacing = 0.015
)
```

**Arguments**

map	A <code>redist_map</code> object
contr	The output of <code>proj_contr()</code> with <code>pfdr=TRUE</code> : A vector containing the contrast and an attribute "q" containing the q-values.
level	The positive false discovery rate level to control.
density	The density of the hatching (roughly what portion is shaded).
spacing	The spacing of the hatches.

**Value**

A `ggplot`.

**Examples**

```
# example code
set.seed(1812)
data(iowa)
map <- redist_map(iowa, existing_plan = cd_2010, pop_tol = 0.01)
plans <- redist_smc(map, 50, silent = TRUE)
plans$dem <- group_frac(map, dem_08, tot_08, plans)

pc = proj_contr(plans, dem, pfdr=TRUE)
redist.plot.contr_pfd(map, pc, level=0.4) # high `level` just to demonstrate
```

---

redist.plot.cores      *Plot Cores*

---

**Description**

Plot Cores

**Usage**

```
redist.plot.cores(shp, plan = NULL, core = NULL, lwd = 2)
```

**Arguments**

shp	A SpatialPolygonsDataFrame or sf object. Required.
plan	A numeric vector with one entry for each precinct in shp. Used to color the districts. Required.
core	Required. integer vector produced by <code>redist.identify.cores()</code> .
lwd	Line width. Defaults to 2.

**Value**

ggplot

---

redist.plot.distr\_qtys

*Plot quantities by district*

---

**Description**

Plots a boxplot of a quantity of interest across districts, with districts optionally sorted by this quantity. Adds reference points for each reference plan, if applicable.

**Usage**

```
redist.plot.distr_qtys(
  plans,
  qty,
  sort = "asc",
  geom = "jitter",
  color_thresh = NULL,
  size = 0.1,
  ref_geom,
  ref_label,
  ...
)
```

**Arguments**

plans	the <code>redist_plans</code> object.
qty	<a href="#">&lt;data-masking&gt;</a> the quantity of interest.
sort	set to "asc" to sort districts in ascending order of qty (the default), "desc" for descending order, or FALSE or "none" for no sorting.
geom	the ggplot2 geom to use in plotting the simulated districts: either "jitter" or "boxplot". Can also take in a function, so long as the function accepts ...
color_thresh	if a number, the threshold to use in coloring the points. Plans with quantities of interest above the threshold will be colored differently than plans below the threshold.

size	The dot size for geom="jitter".
ref_geom	The reference plan geometry type. "line" or "point" can be passed for reasonable defaults. Can also take in a function, so long as the function accepts ...
ref_label	A human-readable name for the reference plan. By default the name in the plan column is used. This can also take in a function which returns a call to ggplot2::labs().
...	passed on to <code>geom_boxplot</code>

**Value**

A ggplot

**Using ggdist**

For custom functions in geom, we can also create more complicated things like rainclouds using the ggdist package. For example:

```
raincloud <- function(...) {
  list(
    ggdist::stat_slab(aes(thickness = ggplot2::after_stat(pdf*n)), scale = 0.7),
    ggdist::stat_dotsinterval(side = "bottom", scale = 0.7,
                             slab_size = NA, quantiles = 200)
  )
}
```

These functions can be then passed to geom.

**Examples**

```
library(dplyr)
data(iowa)

iowa <- redist_map(iowa, existing_plan = cd_2010, pop_tol = 0.05, total_pop = pop)
plans <- redist_smc(iowa, nsims = 100, silent = TRUE)
plans <- plans %>% mutate(pct_dem = group_frac(iowa, dem_08, tot_08))
redist.plot.distr_qtys(plans, pct_dem)

# It also takes custom functions:
redist.plot.distr_qtys(plans, pct_dem, geom = ggplot2::geom_violin)

# With the raincloud example, if you have `ggdist`, you can run:
# redist.plot.distr_qtys(plans, pct_dem, geom = raincloud)

# The reference geom can also be changed via `reg_geom`
r_geom <- function(...) ggplot2::geom_segment(ggplot2::aes(as.integer(.data$.distr_no) - 0.5,
                  xend = as.integer(.data$.distr_no) + 0.5,
                  yend = pct_dem,
                  color = .data$draw),
                  linewidth = 1.2, ...)
```

```
# Finally, the `ref_label` argument can also be swapped for a function, like so:
redist.plot.distr_qtys(plans, pct_dem, geom = ggplot2::geom_violin, ref_geom = r_geom,
  ref_label = function() ggplot2::labs(color = 'Ref.'))
```

---

redist.plot.hist      *Plot a histogram of a summary statistic*

---

### Description

Plots a histogram of a statistic of a [redist\\_plans](#) object, with a reference line for each reference plan, if applicable.

### Usage

```
redist.plot.hist(plans, qty, bins = NULL, ...)
```

```
## S3 method for class 'redist_plans'
hist(x, qty, ...)
```

### Arguments

plans	the redist_plans object.
qty	<data-masking> the statistic.
bins	the number of bins to use in the histogram. Defaults to Freedman-Diaconis rule.
...	passed on to <a href="#">geom_histogram</a>
x	<data-masking> the statistic.

### Value

A ggplot

### Examples

```
library(dplyr)
data(iowa)

iowa <- redist_map(iowa, existing_plan = cd_2010, pop_tol = 0.05)
plans <- redist_smc(iowa, nsims = 100, silent = TRUE)
group_by(plans, draw) %>%
  summarize(pop_dev = max(abs(total_pop/mean(total_pop) - 1))) %>%
  redist.plot.hist(pop_dev)
```

---

redist.plot.majmin      *Majority Minority Plots*

---

### Description

Majority Minority Plots

### Usage

```
redist.plot.majmin(grouppercent, type = "hist", title = "")
```

### Arguments

grouppercent	output from redist.group.percent
type	string in 'hist', 'toptwo', or 'box'
title	ggplot title

### Value

ggplot

---

redist.plot.map      *Plot a Map*

---

### Description

Create a ggplot map. It fills by plan or argument fill. If both are supplied, plan is used as the color and fill as the alpha parameter.

### Usage

```
redist.plot.map(  
  shp,  
  adj,  
  plan = NULL,  
  fill = NULL,  
  fill_label = "",  
  zoom_to = NULL,  
  boundaries = is.null(fill),  
  title = ""  
)
```

**Arguments**

shp	A SpatialPolygonsDataFrame, sf object, or redist_map. Required.
adj	A zero-indexed adjacency list. Created with redist.adjacency if not supplied and needed for coloring. Default is NULL.
plan	<a href="#">&lt;data-masking&gt;</a> A numeric vector with one entry for each precinct in shp. Used to color the districts. Default is NULL. Optional.
fill	<a href="#">&lt;data-masking&gt;</a> A numeric/integer vector with values to color the plot with. Optional.
fill_label	A string title of plot. Defaults to the empty string
zoom_to	<a href="#">&lt;data-masking&gt;</a> An indexing vector of units to zoom the map to.
boundaries	A logical indicating if precinct boundaries should be plotted.
title	A string title of plot. Defaults to empty string. Optional.

**Value**

ggplot map

**Examples**

```
data(iowa)
redist.plot.map(shp = iowa, plan = iowa$cd_2010)

iowa_map <- redist_map(iowa, existing_plan = cd_2010)
redist.plot.map(iowa_map, fill = dem_08/tot_08, zoom_to = (cd_2010 == 1))
```

---

redist.plot.penalty *(Deprecated) Visualize Group Power Penalty*

---

**Description**

Plots the shape of the [add\\_constr\\_grp\\_pow\(\)](#) penalty.

**Usage**

```
redist.plot.penalty(
  tgt_min = 0.55,
  tgt_other = 0.25,
  strength_vra = 2500,
  pow_vra = 1.5,
  limits = TRUE
)
```

**Arguments**

tgt_min	double, defaults to 0.55. The minority target percent.
tgt_other	double, defaults to 0.25. The other group target percent.
strength_vra	double, strength of the VRA constraint.
pow_vra	double, exponent of the VRA constraint.
limits	Whether to limit y axis to 0,500. Default is TRUE for comparability across values.

**Details**

This function allows you to plot the un-exponentiated penalty implemented as `add_constr_grp_pow()`. The function takes two key inputs, `tgt_min` and `tgt_other` which center the minimum penalty spots. A higher y-value indicates a higher penalty and incentivizes moving towards a spot with a lower y-value. The x-axis indicates the group population proportion in a given district.

**Value**

ggplot

---

redist.plot.plans      *Plot a district assignment*

---

**Description**

Plot a district assignment

**Usage**

```
redist.plot.plans(
  plans,
  draws,
  shp,
  qty = NULL,
  interactive = FALSE,
  ...,
  geom = NULL
)
```

**Arguments**

plans	a redist_plans object.
draws	the plan(s) to plot. Will match the draw column of x.
qty	the quantity to plot. Defaults to the district assignment.
interactive	if TRUE, show an interactive map in the viewer rather than a static map. Only uses the first element of draws
...	additional arguments passed to the plotting functions.
geom, shp	the redist_map geometry to use (geom is deprecated).

**Value**

A ggplot

**Examples**

```
library(dplyr)
data(iowa)

iowa <- redist_map(iowa, existing_plan = cd_2010, pop_tol = 0.05, total_pop = pop)
plans <- redist_smc(iowa, nsims = 100, silent = TRUE)
redist.plot.plans(plans, c(1, 2, 3, 4), iowa)
```

---

redist.plot.scatter     *Scatter plot of plan summary statistics*

---

**Description**

Makes a scatterplot of two quantities of interest across districts or plans.

**Usage**

```
redist.plot.scatter(plans, x, y, ..., bigger = TRUE)
```

**Arguments**

plans	the redist_plans object.
x	<a href="#">&lt;data-masking&gt;</a> the quantity to plot on the horizontal axis.
y	<a href="#">&lt;data-masking&gt;</a> the quantity to plot on the vertical axis.
...	passed on to <a href="#">geom_point</a> .
bigger	if TRUE, make the point corresponding to the reference plan larger.

**Value**

A ggplot

**Examples**

```
library(dplyr)
data(iowa)

iowa <- redist_map(iowa, existing_plan = cd_2010, pop_tol = 0.05, total_pop = pop)
plans <- redist_smc(iowa, nsims = 100, silent = TRUE)
plans %>%
  mutate(comp = distr_compactness(iowa)) %>%
  group_by(draw) %>%
  summarize(pop_dev = max(abs(total_pop/mean(total_pop) - 1)),
```

```
comp = comp[1]) %>%
redist.plot.scatter(pop_dev, comp)
```

---

redist.plot.trace      *Make a traceplot for a summary statistic*

---

## Description

For a statistic in a [redist\\_plans](#) object, make a traceplot showing the evolution of the statistic over MCMC iterations.

## Usage

```
redist.plot.trace(plans, qty, district = 1L, ...)
```

## Arguments

plans	the redist_plans object.
qty	<a href="#">&lt;data-masking&gt;</a> the statistic.
district	for redist_plans objects with multiple districts, which district to subset to for plotting. Set to NULL to perform no subsetting.
...	passed on to <a href="#">geom_line</a>

## Value

A ggplot

## Examples

```
library(dplyr)
data(iowa)

iowa_map <- redist_map(iowa, existing_plan = cd_2010, pop_tol = 0.05)
plans <- redist_mergesplit_parallel(iowa_map, nsims = 200, chains = 2, silent = TRUE) %>%
  mutate(dem = group_frac(iowa_map, dem_08, dem_08 + rep_08)) %>%
  number_by(dem)
redist.plot.trace(plans, dem, district = 1)
```

---

redist.plot.varinfo    *Static Variation of Information Plot*

---

**Description**

Static Variation of Information Plot

**Usage**

```
redist.plot.varinfo(plans, group_pop, total_pop, shp)
```

**Arguments**

plans	matrix of district assignments
group_pop	Required Population of subgroup being studied in each precinct.
total_pop	Required. Population of each precinct.
shp	sf dataframe

**Value**

patchworked ggplot

---

redist.plot.wted.adj    *Plot Weighted Border Adjacency*

---

**Description**

Plots the weighted adjacency graph by how often precincts cocur. If an argument to counties is provided, it subsets the edges to plot to those that cross over the county boundary.

**Usage**

```
redist.plot.wted.adj(  
  shp,  
  plans,  
  counties = NULL,  
  ref = TRUE,  
  adj = NULL,  
  plot_shp = TRUE  
)
```

**Arguments**

shp	A SpatialPolygonsDataFrame, sf object, or redist_map. Required.
plans	A redist_plans object or matrix of redistricting plans, where each column indicates a plan and each
counties	unquoted name of a column in shp or a vector of county assignments. Subsets to edges which cross this boundary if supplied.
ref	Plot reference map? Defaults to TRUE which gets the existing plan from
adj	A zero-indexed adjacency list. Extracted from shp if shp is a redist_map. Otherwise created with redist.adjacency if not supplied. Default is NULL.
plot_shp	Should the shapes be plotted? Default is TRUE.

**Value**

ggplot

**Examples**

```
data(iowa)
shp <- redist_map(iowa, existing_plan = cd_2010, pop_tol = 0.01)
plans <- redist_smc(shp, 100)
redist.plot.wted.adj(shp, plans = plans, counties = region)
```

---

redist.prec.pop.overlap

*Compare the Population Overlap Across Plans at the Precinct Level*

---

**Description**

Compare the Population Overlap Across Plans at the Precinct Level

**Usage**

```
redist.prec.pop.overlap(
  plan_old,
  plan_new,
  total_pop,
  weighting = "s",
  normalize = TRUE,
  index_only = FALSE,
  return_mat = FALSE
)
```

**Arguments**

<code>plan_old</code>	The reference plan to compare against
<code>plan_new</code>	The new plan to compare to the reference plan
<code>total_pop</code>	The total population by precinct This can also take a <code>redist_map</code> object and will use the population in that object. If nothing is provided, it weights all entries in plan equally.
<code>weighting</code>	Should weighting be done by sum of populations 's', mean of populations 'm', geometric mean of populations 'g', or none 'n'
<code>normalize</code>	Should entries be normalized by the total population
<code>index_only</code>	Default is FALSE. TRUE returns only one numeric index, the mean of the upper triangle of the matrix, under the weighting and normalization chosen.
<code>return_mat</code>	Defaults to FALSE, where it returns the summary by row. If TRUE returns matrix with <code>length(plan_old)</code> rows and columns. Ignored if <code>index_only = TRUE</code> .

**Value**

numeric vector with `length(plan_old)` entries

**Examples**

```
set.seed(5)
data(iowa)
iowa_map <- redist_map(iowa, total_pop = pop, pop_tol = 0.01, ndists = 4)
plans <- redist_smc(iowa_map, 2, silent = TRUE)
plans_mat <- get_plans_matrix(plans)
ov_vec <- redist_prec_pop_overlap(plans_mat[, 1], plans_mat[, 2], iowa_map)
redist_prec_pop_overlap(plans_mat[, 1], plans_mat[, 2], iowa_map, weighting = "s",
  normalize = FALSE, index_only = TRUE)
```

---

`redist.prep.enumpart` *Prepares a run of the enumpart algorithm by ordering edges*

---

**Description**

Prepares a run of the enumpart algorithm by ordering edges

**Usage**

```
redist.prep.enumpart(
  adj,
  unordered_path,
  ordered_path,
  weight_path = NULL,
  total_pop = NULL
)
```

**Arguments**

adj	zero indexed adjacency list
unordered_path	valid path to output the unordered adjacency map to
ordered_path	valid path to output the ordered adjacency map to
weight_path	A path (not including ".dat") to store a space-delimited file containing a vector of vertex weights. Only supply with total_pop.
total_pop	the vector of precinct populations. Only supply with weight_path

**Value**

0 on success

**References**

Benjamin Fifield, Kosuke Imai, Jun Kawahara, and Christopher T Kenny. "The Essential Role of Empirical Validation in Legislative Redistricting Simulation." Forthcoming, Statistics and Public Policy.

**Examples**

```
## Not run:
temp <- tempdir()
data(fl25)
adj <- redist.adjacency(fl25)
redist.prep.enumpart(adj = adj, unordered_path = paste0(temp, "/unordered"),
  ordered_path = paste0(temp, "/ordered"))

## End(Not run)
```

---

redist.random.subgraph

*Return a random subgraph of a shape*

---

**Description**

random.subgraph returns a random subset of the shp provided

**Usage**

```
redist.random.subgraph(shp, n, adj = NULL)
```

**Arguments**

shp	sf object or SpatialPolygonsDataFrame
n	number of edges to sample. n must be a positive integer.
adj	Optional. zero indexed adjacency list.

**Details**

Snowball sampling with backtracking

**Value**

sf dataframe with n rows

---

redist.read.enumpart *Read Results from enumpart*

---

**Description**

Read Results from enumpart

**Usage**

```
redist.read.enumpart(out_path, skip = 0, n_max = -1L)
```

**Arguments**

out_path	out_path specified in redist.run.enumpart
skip	number of lines to skip
n_max	max number of lines to read

**Value**

district\_membership matrix

**References**

Benjamin Fifield, Kosuke Imai, Jun Kawahara, and Christopher T Kenny. "The Essential Role of Empirical Validation in Legislative Redistricting Simulation." Forthcoming, Statistics and Public Policy.

**Examples**

```
## Not run:  
temp <- tempdir()  
cds <- redist.read.enumpart(out_path = paste0(temp, "/enumerated"))  
  
## End(Not run)
```

---

```
redist.reduce.adjacency
```

*Reduce Adjacency List*

---

**Description**

Tool to help reduce adjacency lists for analyzing subsets of maps.

**Usage**

```
redist.reduce.adjacency(adj, keep_rows)
```

**Arguments**

adj	A zero-indexed adjacency list. Required.
keep_rows	row numbers of precincts to keep

**Value**

zero indexed adjacency list with max value length(keep\_rows) - 1

**Examples**

```
data(f125_adj)
redist.reduce.adjacency(f125_adj, c(2, 3, 4, 6, 21))
```

---

```
redist.reorder
```

*Reorders district numbers*

---

**Description**

Ensures that for each column in the plans object, the first district listed is 1, the second is 2, up to n districts. Assumes that all columns have the same number of districts as the first.

**Usage**

```
redist.reorder(plans)
```

**Arguments**

plans	A numeric vector (if only one map) or matrix with one row for each precinct and one column for each map.
-------	--

**Value**

integer matrix

**Examples**

```

cds <- matrix(c(rep(c(4L, 5L, 2L, 1L, 3L), 5),
  rep(c(5L, 4L, 3L, 2L, 1L), 2), rep(c(4L, 5L, 2L, 1L, 3L), 3)), nrow = 25)
redist.reorder(cds)

```

redist.rsg

*Redistricting via Random Seed and Grow Algorithm***Description**

redist.rsg generates redistricting plans using a random seed a grow algorithm. This is the non-compact districting algorithm described in Chen and Rodden (2013). The algorithm can provide start values for the other redistricting routines in this package.

**Usage**

```
redist.rsg(adj, total_pop, ndists, pop_tol, verbose = TRUE, maxiter = 5000)
```

**Arguments**

adj	List of length N, where N is the number of precincts. Each list element is an integer vector indicating which precincts that precinct is adjacent to. It is assumed that precinct numbers start at 0.
total_pop	numeric vector of length N, where N is the number of precincts. Each element lists the population total of the corresponding precinct, and is used to enforce population constraints.
ndists	integer, the number of districts we want to partition the precincts into.
pop_tol	numeric, indicating how close district population targets have to be to the target population before algorithm converges. thresh=0.05 for example means that all districts must be between 0.95 and 1.05 times the size of target.pop in population size.
verbose	boolean, indicating whether the time to run the algorithm is printed.
maxiter	integer, indicating maximum number of iterations to attempt before convergence to population constraint fails. If it fails once, it will use a different set of start values and try again. If it fails again, redist.rsg() returns an object of all NAs, indicating that use of more iterations may be advised.

**Value**

list, containing three objects containing the completed redistricting plan.

- plan: A vector of length N, indicating the district membership of each precinct.
- district\_list A list of length Ndistrict. Each list contains a vector of the precincts in the respective district.
- district\_pop A vector of length Ndistrict, containing the population totals of the respective districts.

**Author(s)**

Benjamin Fifield, Department of Politics, Princeton University <benfifield@gmail.com>, <https://www.benfifield.com/>

Michael Higgins, Department of Statistics, Kansas State University <mikehiggins@k-state.edu>, <https://www.k-state.edu/stats/about/people/HigginsMichael.html>

Kosuke Imai, Department of Politics, Princeton University <imai@harvard.edu>, <https://imai.fas.harvard.edu>

James Lo, <jameslo@princeton.edu>

Alexander Tarr, Department of Electrical Engineering, Princeton University <atarr@princeton.edu>

**References**

Jowei Chen and Jonathan Rodden (2013) “Unintentional Gerrymandering: Political Geography and Electoral Bias in Legislatures.” *Quarterly Journal of Political Science*. 8(3): 239-269.

**Examples**

```
### Real data example from test set
data(f125)
data(f125_adj)

res <- redist.rsg(adj = f125_adj, total_pop = f125$pop,
  ndists = 3, pop_tol = 0.05)
```

---

redist.run.enumpart    *Runs the enumpart algorithm*

---

**Description**

Runs the enumpart algorithm

**Usage**

```
redist.run.enumpart(
  ordered_path,
  out_path,
  ndists = 2,
  all = TRUE,
  n = NULL,
  weight_path = NULL,
  lower = NULL,
  upper = NULL,
  options = NULL
)
```

**Arguments**

ordered_path	Path used in redist.prep.enumpart (not including ".dat")
out_path	Valid path to output the enumerated districts
ndists	number of districts to enumerate
all	boolean. TRUE outputs all districts. FALSE samples n districts.
n	integer. Number of districts to output if all is FALSE. Returns districts selected from uniform random distribution.
weight_path	A path (not including ".dat") to a space-delimited file containing a vector of vertex weights, to be used along with lower and upper.
lower	A lower bound on each partition's total weight, implemented by rejection sampling.
upper	An upper bound on each partition's total weight.
options	Additional enumpart arguments. Not recommended for use.

**Value**

0 on success

**References**

Benjamin Fifield, Kosuke Imai, Jun Kawahara, and Christopher T Kenny. "The Essential Role of Empirical Validation in Legislative Redistricting Simulation." Forthcoming, Statistics and Public Policy.

**Examples**

```
## Not run:
temp <- tempdir()
redist.run.enumpart(ordered_path = paste0(temp, "/ordered"),
  out_path = paste0(temp, "/enumerated"))

## End(Not run)
```

---

redist.sink.plan      *Sink Plans to 1:ndists*

---

**Description**

Takes a plan and renumbers it to be from 1:ndists

**Usage**

```
redist.sink.plan(plan)
```

**Arguments**

plan                    vector of assignments, required.

**Value**

A vector with an ID that corresponds from 1:ndists, and attribute n indicating the number of districts.

**Examples**

```
data(f125_enum)
plan <- f125_enum$plans[, 5118]
# Subset based on something:
plan <- plan[plan != 2]
plan <- vctrs::vec_group_id(plan)
# Now plan can be used with redist_flip()
plan
```

---

redist.smc\_is\_ci            *(Deprecated) Confidence Intervals for Importance Sampling Estimates*

---

**Description**

Builds a confidence interval for a quantity of interest, given importance sampling weights.

**Usage**

```
redist.smc_is_ci(x, wgt, conf = 0.99)
```

**Arguments**

x                        A numeric vector containing the quantity of interest

wgt                      A numeric vector containing the nonnegative importance weights. Will be normalized automatically.

conf                     The confidence level for the interval.

**Value**

A two-element vector of the form [lower, upper] containing the importance sampling confidence interval.

---

redist.subset	<i>Subset a shp</i>
---------------	---------------------

---

### Description

Subsets a shp object along with its adjacency. Useful for running smaller analyses on pairs of districts. Provide population, ndists, pop\_tol, and sub\_ndists to get proper population parity constraints on subsets.

### Usage

```
redist.subset(shp, adj, keep_rows, total_pop, ndists, pop_tol, sub_ndists)
```

### Arguments

shp	An sf object
adj	A zero-indexed adjacency list. Created with redist.adjacency if not supplied.
keep_rows	row numbers of precincts to keep. Random submap selected if not supplied.
total_pop	numeric vector with one entry for the population of each precinct.
ndists	integer, number of districts in whole map
pop_tol	The strength of the hard population constraint.
sub_ndists	integer, number of districts in subset map

### Value

a list containing the following components:

shp	The subsetted shp object
adj	The subsetted adjacency list for shp
keep_rows	The indices of the rows kept.
sub_ndists	The number of districts in the subset.
sub_pop_tol	The new parity constraint for a subset.

---

redist.uncoarsen      *Uncoarsen a District Matrix*

---

### Description

After a cores analysis or other form of coarsening, sometimes you need to be at the original geography level to be comparable. This takes in a coarsened matrix and uncoarsens it to the original level

### Usage

```
redist.uncoarsen(plans, group_index)
```

### Arguments

plans	A coarsened matrix of plans.
group_index	The index used to coarsen the shape.

### Value

matrix

---

redist.wted.adj      *Create Weighted Adjacency Data*

---

### Description

Create Weighted Adjacency Data

### Usage

```
redist.wted.adj(map = NULL, plans = NULL)
```

### Arguments

map	redist_map
plans	redist_plans

### Value

tibble

### Examples

```
data(iowa)
shp <- redist_map(iowa, existing_plan = cd_2010, pop_tol = 0.01)
plans <- redist_smc(shp, 100)
redist.wted.adj(shp, plans = plans)
```

**Description**

Builds a confidence interval for a quantity of interest. If multiple runs are available, uses the between-run variation to estimate the standard error. If only one run is available, uses information on the SMC particle/plan genealogy to estimate the standard error, using a variant of the method of Olson & Douc (2019). The multiple-run estimator is more reliable, especially for situations with many districts, and should be used when parallelism is available. All reference plans are ignored.

**Usage**

```
redist_ci(plans, x, district = 1L, conf = 0.9, by_chain = FALSE)
```

```
redist_smc_ci(plans, x, district = 1L, conf = 0.9, by_chain = FALSE)
```

```
redist_mcmc_ci(plans, x, district = 1L, conf = 0.9, by_chain = FALSE)
```

**Arguments**

plans	a <a href="#">redist_plans</a> object.
x	the quantity to build an interval for. Tidy-evaluated within plans.
district	for <a href="#">redist_plans</a> objects with multiple districts, which district to subset to. Set to NULL to perform no subsetting.
conf	the desired confidence level.
by_chain	Whether the confidence interval should indicate overall sampling uncertainty (FALSE) or per-chain sampling uncertainty (TRUE). In the latter case the intervals will be wider by a factor of <code>sqrt(runs)</code> .

**Value**

A tibble with three columns: `X`, `X_lower`, and `X_upper`, where `X` is the name of the vector of interest, containing the mean and confidence interval. When used inside [summarize\(\)](#) this will create three columns in the output data.

**Functions**

- `redist_smc_ci()`: Compute confidence intervals for SMC output.
- `redist_mcmc_ci()`: Compute confidence intervals for MCMC output.

## References

- Lee, A., & Whiteley, N. (2018). Variance estimation in the particle filter. *Biometrika*, 105(3), 609-625.
- Olsson, J., & Douc, R. (2019). Numerically stable online estimation of variance in particle filters. *Bernoulli*, 25(2), 1504-1535.
- H. P. Chan and T. L. Lai. A general theory of particle filters in hidden Markov models and some applications. *Ann. Statist.*, 41(6):2877–2904, 2013.

## Examples

```
library(dplyr)
data(iowa)

iowa_map <- redist_map(iowa, existing_plan = cd_2010, pop_tol = 0.05)
plans <- redist_mergesplit_parallel(iowa_map, nsims = 200, chains = 2, silent = TRUE) %>%
  mutate(dem = group_frac(iowa_map, dem_08, dem_08 + rep_08)) %>%
  number_by(dem)
redist_smc_ci(plans, dem)
```

---

redist_constr	<i>Set up constraints for sampling</i>
---------------	--

---

## Description

`redist_constr` objects are used to specify constraints when sampling redistricting plans with `redist_smc()` and `redist_mergesplit()`. Each constraint is specified as a function which scores a given plan. Higher scores are penalized and sampled less frequently.

## Usage

```
redist_constr(map = tibble())
```

## Arguments

`map` a `redist_map()` object; the map that will be used in sampling

## Details

The `redist_constr` object keeps track of sampling constraints in a nested list. You can view the exact structure of this list by calling `str()`. Constraints may be added by using one of the following functions:

- `add_constr_compet()`
- `add_constr_custom()`
- `add_constr_edges_rem()`
- `add_constr_fry_hold()`

- `add_constr_grp_hinge()`
- `add_constr_grp_inv_hinge()`
- `add_constr_grp_pow()`
- `add_constr_incumbency()`
- `add_constr_log_st()`
- `add_constr_multisplits()`
- `add_constr_polsby()`
- `add_constr_pop_dev()`
- `add_constr_segregation()`
- `add_constr_splits()`
- `add_constr_status_quo()`
- `add_constr_total_splits()`

More information about each constraint can be found on the relevant constraint page.

### Value

a `redist_constr` object, which is just a list with a certain nested structure.

### Examples

```
data(iowa)
map_ia <- redist_map(iowa, existing_plan = cd_2010, pop_tol = 0.01)
constr <- redist_constr(map_ia)
constr <- add_constr_splits(constr, strength = 1.5, admin = region)
print(constr)
```

---

redist_flip	<i>'Flip' Markov Chain Monte Carlo Redistricting Simulation (Fifield et al. 2020)</i>
-------------	---

---

### Description

This function allows users to simulate redistricting plans using a Markov Chain Monte Carlo algorithm (Fifield, Higgins, Imai, and Tarr 2020). Several constraints corresponding to substantive requirements in the redistricting process are implemented, including population parity and geographic compactness. In addition, the function includes multiple-swap and simulated tempering functionality to improve the mixing of the Markov Chain.

**Usage**

```

redist_flip(
  map,
  nsims,
  chains = 1,
  warmup = 0,
  init_plan = NULL,
  constraints = add_constr_edges_rem(redist_constr(map), 0.4),
  thin = 1,
  eprob = 0.05,
  lambda = 0,
  temper = FALSE,
  betaseq = "powerlaw",
  betaseqlength = 10,
  betaweights = NULL,
  adapt_lambda = FALSE,
  adapt_eprob = FALSE,
  exact_mh = FALSE,
  adjswaps = TRUE,
  ncores = NULL,
  cl_type = "PSOCK",
  return_all = TRUE,
  init_name = NULL,
  verbose = TRUE
)

```

**Arguments**

map	A <a href="#">redist_map</a> object.
nsims	The number of samples to draw per chain, not including warmup.
chains	The number of parallel chains to run. Each chain will have nsims draws. If <code>init_plan</code> is sampled, each chain will be initialized with its own sampled plan. Defaults to 1.
warmup	The number of warmup samples to discard.
init_plan	The initial state of the map, provided as a single vector to be shared across all chains, or a matrix with chains columns. If not provided, will default to the reference map of the map object, or if none exists, will sample a random initial state using <code>redist_smc</code> . You can also request a random initial state for each chain by setting <code>init_plan="sample"</code> .
constraints	A <code>redist_constr</code> object.
thin	The amount by which to thin the Markov Chain. The default is 1.
eprob	The probability of keeping an edge connected. The default is 0.05.
lambda	lambda The parameter determining the number of swaps to attempt each iteration of the algorithm. The number of swaps each iteration is equal to $\text{Pois}(\text{lambda}) + 1$ . The default is 0.
temper	Whether to use simulated tempering algorithm. Default is FALSE.

betaseq	Sequence of beta values for tempering. The default is powerlaw (see Fifield et. al (2020) for details).
betaseqlength	Length of beta sequence desired for tempering. The default is 10.
betaweights	betaweights Sequence of weights for different values of beta. Allows the user to upweight certain values of beta over others. The default is NULL (equal weighting).
adapt_lambda	adapt_lambda Whether to adaptively tune the lambda parameter so that the Metropolis-Hastings acceptance probability falls between 20% and 40%. Default is FALSE.
adapt_eprob	eprob Whether to adaptively tune the edgcut probability parameter so that the Metropolis-Hastings acceptance probability falls between 20% and 40%. Default is FALSE.
exact_mh	Whether to use the approximate (FALSE) or exact (TRUE) Metropolis-Hastings ratio calculation for accept-reject rule. Default is FALSE.
adjswaps	Flag to restrict swaps of beta so that only values adjacent to current constraint are proposed. The default is TRUE.
ncores	The number of parallel processes to run. Defaults to the number of available cores, capped at the number of chains.
cl_type	The cluster type (see <a href="#">parallel::makeCluster()</a> ). Safest is "PSOCK", but "FORK" may be faster on non-Windows systems.
return_all	If TRUE return all sampled plans; otherwise, just return the final plan from each chain.
init_name	a name for the initial plan, or FALSE to not include the initial plan in the output. Defaults to the column name of the existing plan, or "<init>" if the initial plan is sampled.
verbose	Whether to print initialization statement. Default is TRUE.

## Details

redist\_flip allows for Gibbs constraints to be supplied via a list object passed to constraints. redist\_flip uses a small compactness constraint by default, as this improves the realism of the maps greatly and also leads to large speed improvements. (One of the most time consuming aspects of the flip MCMC backend is checking for district shattering, which is slowed down even further by non-compact districts. As such, it is recommended that all flip simulations use at least a minimal compactness constraint, even if you weaken it from the default settings.) The default is a compact constraint using the edges-removed metric with a weight of 0.6. For very small maps (< 100 precincts), you will likely want to weaken (lower) this constraint, while for very large maps (> 5000 precincts), you will likely want to strengthen (increase) this constraint. Otherwise, for most maps, the default constraint should be a good starting place.

redist\_flip samples from a known target distribution which can be described using the constraints. The following describes the constraints available. The general advice is to set weights in a way that gets between 20% and 40% acceptance on average, though more tuning advice is available in the vignette on using MCMC methods. Having too small of an acceptance rate indicates that the weights within constraints are too large and will impact sampling efficiency. If the Metropolis Hastings

acceptance rate is too large, this may impact the target distribution, but may be fine for general exploration of possible maps.

There are currently 9 implemented constraint types, though `compact` and `partisan` have sub-types which are specified via a character `metric` within their respective list objects. The constraints are as follows:

- `compact` - biases the algorithm towards drawing more compact districts.
- `weight` - the coefficient to put on the Gibbs constraint
- `metric` - which metric to use. Must be one of `edges-removed` (the default), `polsby-popper`, `fryer-holden`, or `log-st`. Using `Polsby Popper` is generally not recommended, as `edges-removed` is faster and highly correlated. `log-st` can be used to match the target distribution of `redist_smc` or `redist_mergesplit`.
- `areas` - Only used with `polsby-popper` - A vector of precinct areas.
- `borderlength_mat` - Only used with `polsby-popper` - A matrix of precinct border lengths.
- `ssdmat` - Only used with `fryer-holden` - A matrix of squared distances between precinct centroids.
- `ssd_denom` - Only used with `fryer-holden` - a positive integer to use as the normalizing constant for the Relative Proximity Index.
- `population` - A Gibbs constraint to complement the hard population constraint set by `pop_tol`. This penalizes moves which move away from smaller population parity deviations. It is very useful when an `init_plan` sits outside of the desired `pop_tol` but there are substantive reasons to use that plan. This constraint uses the input to `total_pop`.
- `weight` - the coefficient to put on the Gibbs constraint
- `countysplit` This is a Gibbs constraint to minimize county splits. Unlike SMC's county constraint, this allows for more than `ndists - 1` splits and does not require that counties are contiguous.
- `weight` - the coefficient to put on the Gibbs constraint
- `hinge` This uses the proportion of a `group` in a district and matches to the nearest target proportion, and then creates a penalty of  $\sqrt{\max(0, \text{nearest.target} - \text{group.pct})}$ .
- `weight` - the coefficient to put on the Gibbs constraint
- `minorityprop` - A numeric vector of minority proportions (between 0 and 1) which districts should aim to have
- `vra` This takes two target proportions of the presence of a minority group within a district.  $(|\text{target.min} - \text{group.pct}| |\text{target.other} - \text{group.pct}|)^{1.5}$
- `weight` - the coefficient to put on the Gibbs constraint
- `target_min` - the target minority percentage. Often, this is set to 0.55 to encourage minority majority districts.
- `target_other` - the target minority percentage for non majority minority districts.
- `minority` This constraint sorts the districts by the proportion of a `group` in a district and compares the highest districts to the entries of `minorityprop`. This takes the form  $\sum_{i=1}^n \sqrt{|\text{group.pct}(i) - \text{minorityprop}(i)|}$  where `n` is the length of `minorityprop` input.
- `weight` - the coefficient to put on the Gibbs constraint

- `minorityprop` - A numeric vector of minority proportions (between 0 and 1) which districts should aim to have
- `similarity` This is a status-quo constraint which penalizes plans which are very different from the starting place. It is useful for local exploration.
- `weight` - the coefficient to put on the Gibbs constraint
- `partisan` This is a constraint which minimizes partisan bias, either as measured as the difference from proportional representation or as the magnitude of the efficiency gap.
- `weight` - the coefficient to put on the Gibbs constraint
- `rvote` - An integer vector of votes for Republicans or other party
- `dvote` - An integer vector of votes for Democrats or other party
- `metric` - which metric to use. Must be one of `proportional-representation` or `efficiency-gap`.
- `segregation` This constraint attempts to minimize the degree of dissimilarity between districts by group population.
- `weight` - the coefficient to put on the Gibbs constraint

### Value

A `redist_plans` object containing the simulated plans. If `chains > 1`, the output will include a chain column indicating which chain each plan came from.

### References

Fifield, B., Higgins, M., Imai, K., & Tarr, A. (2020). Automated redistricting simulation using Markov chain Monte Carlo. *Journal of Computational and Graphical Statistics*, 29(4), 715-728.

### Examples

```
data(iowa)
iowa_map <- redist_map(iowa, ndists = 4, existing_plan = cd_2010, total_pop = pop,
  pop_tol = 0.05)
sims <- redist_flip(map = iowa_map, nsims = 100)

# Multiple chains for convergence diagnostics
sims_chains <- redist_flip(iowa_map, nsims = 20, chains = 2, ncores = 2)
```

---

redist\_flip\_anneal      *Flip MCMC Redistricting Simulator using Simulated Annealing*

---

### Description

`redist_flip_anneal` simulates congressional redistricting plans using Markov chain Monte Carlo methods coupled with simulated annealing.

**Usage**

```

redist_flip_anneal(
  map,
  nsims,
  warmup = 0,
  init_plan = NULL,
  constraints = redist_constr(),
  num_hot_steps = 40000,
  num_annealing_steps = 60000,
  num_cold_steps = 20000,
  eprob = 0.05,
  lambda = 0,
  adapt_lambda = FALSE,
  adapt_eprob = FALSE,
  exact_mh = FALSE,
  maxiterrsg = 5000,
  verbose = TRUE
)

```

**Arguments**

map	A <a href="#">redist_map</a> object.
nsims	The number of samples to draw, not including warmup.
warmup	The number of warmup samples to discard.
init_plan	A vector containing the congressional district labels of each geographic unit. The default is NULL. If not provided, a random initial plan will be generated using <code>redist_smc</code> . You can also request to initialize using <code>redist.rsg</code> by supplying 'rsg', though this is not recommended behavior.
constraints	A <code>redist_constr</code> object.
num_hot_steps	The number of steps to run the simulator at $\beta = 0$ . Default is 40000.
num_annealing_steps	The number of steps to run the simulator with linearly changing beta schedule. Default is 60000
num_cold_steps	The number of steps to run the simulator at $\beta = 1$ . Default is 20000.
eprob	The probability of keeping an edge connected. The default is 0.05.
lambda	The parameter determining the number of swaps to attempt each iteration of the algorithm. The number of swaps each iteration is equal to $\text{Pois}(\lambda) + 1$ . The default is 0.
adapt_lambda	Whether to adaptively tune the lambda parameter so that the Metropolis-Hastings acceptance probability falls between 20% and 40%. Default is FALSE.
adapt_eprob	Whether to adaptively tune the edgcut probability parameter so that the Metropolis-Hastings acceptance probability falls between 20% and 40%. Default is FALSE.
exact_mh	Whether to use the approximate (0) or exact (1) Metropolis-Hastings ratio calculation for accept-reject rule. Default is FALSE.

maxiterrsg	Maximum number of iterations for random seed-and-grow algorithm to generate starting values. Default is 5000.
verbose	Whether to print initialization statement. Default is TRUE.

**Value**

redist\_plans

---

redist_map	<i>Create a redist_map object.</i>
------------	------------------------------------

---

**Description**

Sets up a redistricting problem.

**Usage**

```
redist_map(
  ...,
  existing_plan = NULL,
  pop_tol = NULL,
  total_pop = c("pop", "population", "total_pop", "POP100"),
  ndists = NULL,
  pop_bounds = NULL,
  adj = NULL,
  adj_col = "adj",
  planarize = 3857
)

as_redist_map(x)
```

**Arguments**

...	column elements to be bound into a redist_map object or a single list or data.frame. These will be passed on to the <a href="#">tibble::tibble</a> constructor.
existing_plan	<a href="#">&lt;tidy-select&gt;</a> the existing district assignment. Must be numeric or convertible to numeric.
pop_tol	<a href="#">&lt;data-masking&gt;</a> the population tolerance. The percentage deviation from the average population will be constrained to be no more than this number. If existing_plan is provided, defaults to the parity of that plan; otherwise, defaults to 0.01.
total_pop	<a href="#">&lt;tidy-select&gt;</a> the vector of precinct populations. Defaults to the pop, population, or total_pop columns, if one exists.
ndists	<a href="#">&lt;data-masking&gt;</a> the integer number of districts to partition the map into. Must be specified if existing_plan is not supplied.

pop_bounds	<data-masking> more specific population bounds, in the form of <code>c(lower, target, upper)</code> .
adj	the adjacency graph for the object. Defaults to being computed from the data if it is coercible to a shapefile.
adj_col	the name of the adjacency graph column
planarize	a number, indicating the CRS to project the shapefile to if it is latitude-longitude based. Set to NULL or FALSE to avoid planarizing.
x	an object to be coerced

### Details

A `redist_map` object is a `tibble::tibble` which contains an adjacency list and additional information about the number of districts and population bounds. It supports all of the `dplyr` generics, and will adjust the adjacency list and attributes according to these functions; i.e., if we filter to a subset of units, the graph will change to subset to these units, and the population bounds will adjust accordingly. If an existing map is also attached to the object, the number of districts will also adjust. Subsetting with ``[`` and ``[[`` does not recompute graphs or attributes.

Other useful methods for `redist_map` objects:

- `merge_by`
- `get_adj`
- `plot.redist_map`

### Value

A `redist_map` object

### Examples

```
data(f125)
d <- redist_map(f125, ndists = 3, pop_tol = 0.05, total_pop = pop)
dplyr::filter(d, pop >= 10e3)
```

---

redist_mergesplit	<i>Merge-Split/Recombination MCMC Redistricting Sampler (Carter et al. 2019)</i>
-------------------	--

---

### Description

`redist_mergesplit` uses a Markov Chain Monte Carlo algorithm (Carter et al. 2019; based on DeFord et. al 2019) to generate congressional or legislative redistricting plans according to contiguity, population, compactness, and administrative boundary constraints. The MCMC proposal is the same as is used in the SMC sampler (McCartan and Imai 2023); it is similar but not identical to those used in the references. 1-level hierarchical Merge-split is supported through the `counties` parameter; unlike in the SMC algorithm, this does not guarantee a maximum number of county splits.

**Usage**

```

redist_mergesplit(
  map,
  nsims,
  warmup = if (is.null(init_plan)) 10 else max(100, nsims%%5),
  thin = 1L,
  init_plan = NULL,
  counties = NULL,
  compactness = 1,
  constraints = list(),
  constraint_fn = function(m) rep(0, ncol(m)),
  adapt_k_thresh = 0.99,
  k = NULL,
  init_name = NULL,
  silly_adj_fix = FALSE,
  verbose = FALSE,
  silent = FALSE
)

```

**Arguments**

map	A <a href="#">redist_map</a> object.
nsims	The number of samples to draw, including warmup.
warmup	The number of warmup samples to discard. Recommended to be at least the first 20% of samples, and in any case no less than around 100 samples, unless initializing from a random plan.
thin	Save every thin-th sample. Defaults to no thinning (1).
init_plan	The initial state of the map. If not provided, will default to the reference map of the map object, or if none exists, will sample a random initial state using <a href="#">redist_smc</a> . You can also request a random initial state by setting <code>init_plan="sample"</code> .
counties	A vector containing county (or other administrative or geographic unit) labels for each unit, which may be integers ranging from 1 to the number of counties, or a factor or character vector. If provided, the algorithm will generate maps tend to follow county lines. There is no strength parameter associated with this constraint. To adjust the number of county splits further, or to constrain a second type of administrative split, consider using <code>add_constr_splits()</code> , <code>add_constr_multisplits()</code> , and <code>add_constr_total_splits()</code> .
compactness	Controls the compactness of the generated districts, with higher values preferring more compact districts. Must be nonnegative. See the 'Details' section for more information, and computational considerations.
constraints	A list containing information on constraints to implement. See the 'Details' section for more information.
constraint_fn	A function which takes in a matrix where each column is a redistricting plan and outputs a vector of log-weights, which will be added to the final weights.

adapt_k_thresh	The threshold value used in the heuristic to select a value $k_i$ for each splitting iteration. Set to 0.9999 or 1 if the algorithm does not appear to be sampling from the target distribution. Must be between 0 and 1.
k	The number of edges to consider cutting after drawing a spanning tree. Should be selected automatically in nearly all cases.
init_name	a name for the initial plan, or FALSE to not include the initial plan in the output. Defaults to the column name of the existing plan, or "<init>" if the initial plan is sampled.
silly_adj_fix	Heuristic for fixing weird inputs.
verbose	Whether to print out intermediate information while sampling. Recommended.
silent	Whether to suppress all diagnostic information.

### Details

This function draws samples from a specific target measure, controlled by the map, compactness, and constraints parameters.

Key to ensuring good performance is monitoring the acceptance rate, which is reported at the sample level in the output. Users should also check diagnostics of the sample by running `summary.redist_plans()`.

Higher values of compactness sample more compact districts; setting this parameter to 1 is computationally efficient and generates nicely compact districts.

### Value

`redist_mergesplit` returns an object of class `redist_plans` containing the simulated plans.

### References

Carter, D., Herschlag, G., Hunter, Z., and Mattingly, J. (2019). A merge-split proposal for reversible Monte Carlo Markov chain sampling of redistricting plans. arXiv preprint arXiv:1911.01503.

McCartan, C., & Imai, K. (2023). Sequential Monte Carlo for Sampling Balanced and Compact Redistricting Plans. *Annals of Applied Statistics* 17(4). Available at [doi:10.1214/23AOAS1763](https://doi.org/10.1214/23AOAS1763).

DeFord, D., Duchin, M., and Solomon, J. (2019). Recombination: A family of Markov chains for redistricting. arXiv preprint arXiv:1911.05725.

### Examples

```
data(fl25)

fl_map <- redist_map(fl25, ndists = 3, pop_tol = 0.1)

sampled_basic <- redist_mergesplit(fl_map, 10000)

sampled_constr <- redist_mergesplit(fl_map, 10000, constraints = list(
  incumbency = list(strength = 1000, incumbents = c(3, 6, 25))
))
```

---

 redist\_mergesplit\_parallel

*Parallel Merge-Split/Recombination MCMC Redistricting Sampler*


---

## Description

redist\_mergesplit\_parallel() runs redist\_mergesplit() on several chains in parallel.

## Usage

```
redist_mergesplit_parallel(
  map,
  nsims,
  chains = 1,
  warmup = if (is.null(init_plan)) 10 else max(100, nsims%%5),
  thin = 1L,
  init_plan = NULL,
  counties = NULL,
  compactness = 1,
  constraints = list(),
  constraint_fn = function(m) rep(0, ncol(m)),
  adapt_k_thresh = 0.99,
  k = NULL,
  ncores = NULL,
  cl_type = "PSOCK",
  return_all = TRUE,
  init_name = NULL,
  silly_adj_fix = FALSE,
  verbose = FALSE,
  silent = FALSE
)
```

## Arguments

map	A <a href="#">redist_map</a> object.
nsims	The number of samples to draw, including warmup.
chains	the number of parallel chains to run. Each chain will have nsims draws. If init_plan is sampled, each chain will be initialized with its own sampled plan.
warmup	The number of warmup samples to discard. Recommended to be at least the first 20% of samples, and in any case no less than around 100 samples, unless initializing from a random plan.
thin	Save every thin-th sample. Defaults to no thinning (1).
init_plan	The initial state of the map, provided as a single vector to be shared across all chains, or a matrix with chains columns. If not provided, will default to the reference map of the map object, or if none exists, will sample a random initial

	state using <code>redist_smc</code> . You can also request a random initial state for each chain by setting <code>init_plan="sample"</code> .
<code>counties</code>	A vector containing county (or other administrative or geographic unit) labels for each unit, which may be integers ranging from 1 to the number of counties, or a factor or character vector. If provided, the algorithm will generate maps tend to follow county lines. There is no strength parameter associated with this constraint. To adjust the number of county splits further, or to constrain a second type of administrative split, consider using <code>add_constr_splits()</code> , <code>add_constr_multisplits()</code> , and <code>add_constr_total_splits()</code> .
<code>compactness</code>	Controls the compactness of the generated districts, with higher values preferring more compact districts. Must be nonnegative. See the 'Details' section for more information, and computational considerations.
<code>constraints</code>	A list containing information on constraints to implement. See the 'Details' section for more information.
<code>constraint_fn</code>	A function which takes in a matrix where each column is a redistricting plan and outputs a vector of log-weights, which will be added to the final weights.
<code>adapt_k_thresh</code>	The threshold value used in the heuristic to select a value $k_i$ for each splitting iteration. Set to 0.9999 or 1 if the algorithm does not appear to be sampling from the target distribution. Must be between 0 and 1.
<code>k</code>	The number of edges to consider cutting after drawing a spanning tree. Should be selected automatically in nearly all cases.
<code>ncores</code>	the number of parallel processes to run. Defaults to the maximum available.
<code>cl_type</code>	the cluster type (see <code>makeCluster()</code> ). Safest is "PSOCK", but "FORK" may be appropriate in some settings.
<code>return_all</code>	if TRUE return all sampled plans; otherwise, just return the final plan from each chain.
<code>init_name</code>	a name for the initial plan, or FALSE to not include the initial plan in the output. Defaults to the column name of the existing plan, or "<init>" if the initial plan is sampled.
<code>silly_adj_fix</code>	Heuristic for fixing weird inputs.
<code>verbose</code>	Whether to print out intermediate information while sampling. Recommended.
<code>silent</code>	Whether to suppress all diagnostic information.

### Details

This function draws samples from a specific target measure, controlled by the `map`, `compactness`, and `constraints` parameters.

Key to ensuring good performance is monitoring the acceptance rate, which is reported at the sample level in the output. Users should also check diagnostics of the sample by running `summary.redist_plans()`.

Higher values of `compactness` sample more compact districts; setting this parameter to 1 is computationally efficient and generates nicely compact districts.

### Value

A `redist_plans` object with all of the simulated plans, and an additional chain column indicating the chain the plan was drawn from.

## References

Carter, D., Herschlag, G., Hunter, Z., and Mattingly, J. (2019). A merge-split proposal for reversible Monte Carlo Markov chain sampling of redistricting plans. arXiv preprint arXiv:1911.01503.

McCartan, C., & Imai, K. (2023). Sequential Monte Carlo for Sampling Balanced and Compact Redistricting Plans. *Annals of Applied Statistics* 17(4). Available at [doi:10.1214/23AOAS1763](https://doi.org/10.1214/23AOAS1763).

DeFord, D., Duchin, M., and Solomon, J. (2019). Recombination: A family of Markov chains for redistricting. arXiv preprint arXiv:1911.05725.

## Examples

```
## Not run:
data(f125)
fl_map <- redist_map(f125, ndists = 3, pop_tol = 0.1)
sampled <- redist_mergesplit_parallel(fl_map, nsims = 100, chains = 100)

## End(Not run)
```

---

redist\_plans

*A set of redistricting plans*

---

## Description

A `redist_plans` object is essentially a data frame of summary information on each district and each plan, along with the matrix of district assignments and information about the simulation process used to generate the plans.

## Usage

```
redist_plans(plans, map, algorithm, wgt = NULL, ...)
```

## Arguments

<code>plans</code>	a matrix with <code>n_precinct</code> columns and <code>n_sims</code> rows, or a single vector of precinct assignments.
<code>map</code>	a <code>redist_map</code> object
<code>algorithm</code>	the algorithm used to generate the plans (usually "smc" or "mcmc")
<code>wgt</code>	the weights to use, if any.
<code>...</code>	Other named attributes to set

## Details

The first two columns of the data frame will be `draw`, a factor indexing the simulation draw, and `district`, an integer indexing the districts within a plan. The data frame will therefore have `n_sims*ndists` rows. As a data frame, the usual `dplyr` methods will work.

Other useful methods for `redist_plans` objects:

- `summary.redist_plans`
- `add_reference`
- `subset_sampled`
- `subset_ref`
- `pullback`
- `number_by`
- `match_numbers`
- `is_county_split`
- `prec_assignment`
- `plan_distances`
- `get_plans_matrix`
- `get_plans_weights`
- `get_sampling_info`
- `as.matrix.redist_plans`
- `plot.redist_plans`

## Value

a new `redist_plans` object.

## Examples

```
data(iowa)
```

```
iowa <- redist_map(iowa, existing_plan = cd_2010, pop_tol = 0.05, total_pop = pop)
rsg_plan <- redist.rsg(iowa$adj, iowa$pop, ndists = 4, pop_tol = 0.05)$plan
redist_plans(rsg_plan, iowa, "rsg")
```

---

redist\_quantile\_trunc *Helper function to truncate importance weights*

---

**Description**

Defined as  $\text{pmin}(x, \text{quantile}(x, 1 - \text{length}(x)^{-0.5}))$

**Usage**

```
redist_quantile_trunc(x)
```

**Arguments**

x                    the weights

**Value**

numeric vector

**Examples**

```
redist_quantile_trunc(c(1, 2, 3, 4))
```

---

redist\_shortburst        *Redistricting Optimization through Short Bursts*

---

**Description**

This function uses [redist\\_mergesplit\(\)](#) or [redist\\_flip\(\)](#) to optimize a redistrict plan according to a user-provided criteria. It does so by running the Markov chain for "short bursts" of usually 10 iterations, and then starting the chain anew from the best plan in the burst, according to the criteria. This implements the ideas in the below-referenced paper, "Voting Rights, Markov Chains, and Optimization by Short Bursts."

**Usage**

```
redist_shortburst(  
  map,  
  score_fn = NULL,  
  stop_at = NULL,  
  burst_size = ifelse(backend == "mergesplit", 10L, 50L),  
  max_bursts = 500L,  
  maximize = TRUE,  
  init_plan = NULL,  
  counties = NULL,
```

```

constraints = redist_constr(map),
compactness = 1,
adapt_k_thresh = 0.95,
reversible = TRUE,
fixed_k = NULL,
return_all = TRUE,
thin = 1L,
backend = "mergesplit",
flip_lambda = 0,
flip_eprob = 0.05,
verbose = TRUE
)

```

### Arguments

map	A <a href="#">redist_map</a> object.
score_fn	A function which takes a matrix of plans and returns a score (or, generally, a row vector) for each plan. Can also be a purrr-style anonymous function. See <a href="#">?scorers</a> for some function factories for common scoring rules.
stop_at	A threshold to stop optimization at. When <code>score_fn</code> returns a row vector per plan, <code>maximize</code> can be an equal-length vector specifying a threshold for each dimension, which must all be met for the algorithm to stop.
burst_size	The size of each burst. 10 is recommended for the <code>mergesplit</code> backend and 50 for the <code>flip</code> backend. Can also provide burst schedule function which takes the current iteration (an integer) and returns the desired burst size. This can be a random function.
max_bursts	The maximum number of bursts to run before returning.
maximize	If TRUE, try to maximize the score; otherwise, try to minimize it. When <code>score_fn</code> returns a row vector per plan, <code>maximize</code> can be an equal-length vector specifying whether each dimension should be maximized or minimized.
init_plan	The initial state of the map. If not provided, will default to the reference map of the map object, or if none exists, will sample a random initial state using <a href="#">redist_smc()</a> . You can also request a random initial state by setting <code>init_plan="sample"</code> .
counties	A vector containing county (or other administrative or geographic unit) labels for each unit, which may be integers ranging from 1 to the number of counties, or a factor or character vector. If provided, the algorithm will only generate maps which split up to <code>ndists-1</code> counties. If no county-split constraint is desired, this parameter should be left blank.
constraints	A <code>redist_constr</code> with Gibbs constraints.
compactness	Controls the compactness of the generated districts, with higher values preferring more compact districts. Must be non-negative. See <a href="#">redist_mergesplit</a> for more information.
adapt_k_thresh	The threshold value used in the heuristic to select a value <code>k_i</code> for each splitting iteration.
reversible	If FALSE and <code>backend="mergesplit"</code> , the Markov chain used will not be reversible. This may speed up optimization.

fixed_k	If not NULL, will be used to set the k parameter for the mergesplit backend. If e.g. k=1 then the best edge in each spanning tree will be used. Lower values may speed up optimization at the cost of the Markov chain no longer targeting a known distribution. Recommended only in conjunction with reversible=FALSE.
return_all	Whether to return all the burst results or just the best one (generally, the Pareto frontier). Recommended for monitoring purposes.
thin	Save every thin-th sample. Defaults to no thinning (1). Ignored if return_all=TRUE.
backend	the MCMC algorithm to use within each burst, either "mergesplit" or "flip".
flip_lambda	The parameter determining the number of swaps to attempt each iteration of flip mcmc. The number of swaps each iteration is equal to $\text{Pois}(\text{lambda}) + 1$ . The default is 0.
flip_eprob	The probability of keeping an edge connected in flip mcmc. The default is 0.05.
verbose	Whether to print out intermediate information while sampling. Recommended for monitoring purposes.

### Value

a redist\_plans object containing the final best plan (or the best plans after each burst, if return\_all=TRUE).

### References

Cannon, S., Goldbloom-Helzner, A., Gupta, V., Matthews, J. N., & Suwal, B. (2020). Voting Rights, Markov Chains, and Optimization by Short Bursts. *arXiv preprint arXiv:2011.02288*.

### Examples

```
data(iowa)

iowa_map <- redist_map(iowa, existing_plan = cd_2010, pop_tol = 0.01)
redist_shortburst(iowa_map, scorer_frac_kept(iowa_map), max_bursts = 50)
redist_shortburst(iowa_map, ~ 1 - scorer_frac_kept(iowa_map)(.), max_bursts = 50)
```

---

redist\_smc

*SMC Redistricting Sampler (McCartan and Imai 2023)*


---

### Description

redist\_smc uses a Sequential Monte Carlo algorithm (McCartan and Imai 2023) to generate representative samples of congressional or legislative redistricting plans according to contiguity, population, compactness, and administrative boundary constraints.

**Usage**

```

redist_smc(
  map,
  nsims,
  counties = NULL,
  compactness = 1,
  constraints = list(),
  resample = TRUE,
  runs = 1L,
  ncores = 0L,
  init_particles = NULL,
  n_steps = NULL,
  adapt_k_thresh = 0.99,
  seq_alpha = 0.5,
  truncate = (compactness != 1),
  trunc_fn = redist_quantile_trunc,
  pop_temper = 0,
  final_infl = 1,
  ref_name = NULL,
  verbose = FALSE,
  silent = FALSE
)

```

**Arguments**

map	A <a href="#">redist_map()</a> object.
nsims	The number of samples to draw.
counties	A vector containing county (or other administrative or geographic unit) labels for each unit, which may be integers ranging from 1 to the number of counties, or a factor or character vector. If provided, the algorithm will only generate maps which split up to <code>ndists-1</code> counties. Even there are fewer counties than <code>ndists - 1</code> , the spanning trees will change the results of the simulations. There is no strength parameter associated with this constraint. To adjust the number of county splits further, or to constrain a second type of administrative split, consider using <a href="#">add_constr_splits()</a> , <a href="#">add_constr_multisplits()</a> , and <a href="#">add_constr_total_splits()</a> .
compactness	Controls the compactness of the generated districts, with higher values preferring more compact districts. Must be nonnegative. See the 'Details' section for more information, and computational considerations.
constraints	A <a href="#">redist_constr()</a> object or a list containing information on sampling constraints. See <a href="#">constraints</a> for more information.
resample	Whether to perform a final resampling step so that the generated plans can be used immediately. Set this to <code>FALSE</code> to perform direct importance sampling estimates, or to adjust the weights manually.
runs	How many independent parallel runs to conduct. Each run will have <code>nsims</code> simulations. Multiple runs allows for estimation of simulation standard errors. Output will only be shown for the first run. For compatibility with MCMC methods, runs are identified with the <code>chain</code> column in the output.

ncores	How many cores to use to parallelize plan generation within each run. The default, 0, will use the number of available cores on the machine as long as nsims and the number of units is large enough. If runs>1 you will need to set this manually. If more than one core is used, the sampler output will not be fully reproducible with <code>set.seed()</code> . If full reproducibility is desired, set ncores=1.
init_particles	A matrix of partial plans to begin sampling from. For advanced use only. The matrix must have nsims columns and a row for every precinct. It is important to ensure that the existing districts meet contiguity and population constraints, or there may be major issues when sampling.
n_steps	How many steps to run the SMC algorithm for. Each step splits off a new district. Defaults to all remaining districts. If fewer than the number of remaining splits, reference plans are disabled.
adapt_k_thresh	The threshold value used in the heuristic to select a value $k_i$ for each splitting iteration. Higher values are more accurate but may require more computation. Set to 1 for the most conservative sampling. Must be between 0 and 1.
seq_alpha	The amount to adjust the weights by at each resampling step; higher values prefer exploitation, while lower values prefer exploration. Must be between 0 and 1.
truncate	Whether to truncate the importance sampling weights at the final step by <code>trunc_fn</code> . Recommended if <code>compactness</code> is not 1. Truncation only applied if <code>resample=TRUE</code> .
trunc_fn	A function which takes in a vector of weights and returns a truncated vector. If the <code>loo</code> package is installed (strongly recommended), will default to Pareto-smoothed Importance Sampling (PSIS) rather than naive truncation.
pop_temper	The strength of the automatic population tempering. Try values of 0.01-0.05 to start if the algorithm gets stuck on the final few splits.
final_infl	A multiplier for the population constraint on the final iteration. Used to loosen the constraint when the sampler is getting stuck on the final split. <code>pop_temper</code> should be tried first, since using <code>final_infl</code> will actually change the target distribution.
ref_name	a name for the existing plan, which will be added as a reference plan, or FALSE to not include the initial plan in the output. Defaults to the column name of the existing plan.
verbose	Whether to print out intermediate information while sampling. Recommended.
silent	Whether to suppress all diagnostic information.

## Details

This function draws samples from a specific target measure controlled by the `map`, `compactness`, and `constraints` parameters.

Key to ensuring good performance is monitoring the efficiency of the resampling process at each SMC stage. Unless `silent=FALSE`, this function will print out the effective sample size of each resampling step to allow the user to monitor the efficiency. If `verbose=TRUE` the function will also print out information on the  $k_i$  values automatically chosen and the acceptance rate (based on the population constraint) at each step. Users should also check diagnostics of the sample by running `summary.redist_plans()`.

Higher values of compactness sample more compact districts; setting this parameter to 1 is computationally efficient and generates nicely compact districts. Values of other than 1 may lead to highly variable importance sampling weights. In these cases, these weights are by default truncated using `redist_quantile_trunc()` to stabilize the resulting estimates, but if truncation is used, a specific truncation function should probably be chosen by the user.

### Value

`redist_smc` returns a `redist_plans` object containing the simulated plans.

### References

McCartan, C., & Imai, K. (2023). Sequential Monte Carlo for Sampling Balanced and Compact Redistricting Plans. *Annals of Applied Statistics* 17(4). Available at [doi:10.1214/23AOAS1763](https://doi.org/10.1214/23AOAS1763).

### Examples

```
data(fl25)

fl_map <- redist_map(fl25, ndists = 3, pop_tol = 0.1)

sampled_basic <- redist_smc(fl_map, 5000)

constr <- redist_constr(fl_map)
constr <- add_constr_incumbency(constr, strength = 100, incumbents = c(3, 6, 25))
sampled_constr <- redist_smc(fl_map, 5000, constraints = constr)

# Multiple parallel independent runs
redist_smc(fl_map, 1000, runs = 2)

# One run with multiple cores
redist_smc(fl_map, 1000, ncores = 2)
```

---

scorer-arith

*Scoring function arithmetic*

---

### Description

`redist_scorer` functions may be multiplied by constants and/or added together to form linear combinations.

### Usage

```
## S3 method for class 'redist_scorer'
x * fn2

## S3 method for class 'redist_scorer'
```

```
fn1 + fn2

## S3 method for class 'redist_scorer'
fn1 - fn2
```

### Arguments

x                    a numeric or a redist\_scorer function, from [scorers](#)  
 fn2                  a redist\_scorer function, from [scorers](#)  
 fn1                  a redist\_scorer function, from [scorers](#)

### Value

function of class redist\_scorer

---

scorer-combine	<i>Combine scoring functions</i>
----------------	----------------------------------

---

### Description

redist\_scorer functions may be combined together to optimize along multiple dimensions. Rather than linearly combining multiple scorers to form a single objective as with [scorer-arith](#), these functions allow analysts to approximate the Pareto frontier for a set of scorers.

### Usage

```
combine_scorers(...)
```

```
## S3 method for class 'redist_scorer'
cbind(..., deparse.level = 1)
```

### Arguments

...                    a numeric or a redist\_scorer function, from [scorers](#)  
 deparse.level    As in [cbind\(\)](#).

### Value

function of class redist\_scorer. Will return a matrix with each column containing every plan's scores for a particular scoring function.

---

scorer\_group\_pct      *Scoring functions for redist\_shortburst*

---

### Description

The output of these functions may be passed into `redist_shortburst()` as `score_fn`. Scoring functions have type `redist_scorer` and may be combined together using basic arithmetic operations.

### Usage

```
scorer_group_pct(map, group_pop, total_pop, k = 1)

scorer_pop_dev(map)

scorer_splits(map, counties)

scorer_multisplits(map, counties)

scorer_frac_kept(map)

scorer_polsby_popper(map, perim_df = NULL, areas = NULL, m = 1)

scorer_status_quo(map, existing_plan = get_existing(map))
```

### Arguments

<code>map</code>	A <a href="#">redist_map</a> object.
<code>group_pop</code>	A numeric vector with the population of the group for every precinct.
<code>total_pop</code>	A numeric vector with the population for every precinct.
<code>k</code>	the k-th from the top group fraction to return as the score.
<code>counties</code>	A numeric vector with an integer from 1:n_counties
<code>perim_df</code>	perimeter distance dataframe from <a href="#">redistmetrics::prep_perims()</a>
<code>areas</code>	area of each precinct (ie <code>st_area(map)</code> )
<code>m</code>	the m-th from the bottom Polsby Popper to return as the score. Defaults to 1, the minimum Polsby Popper score
<code>existing_plan</code>	A vector containing the current plan.

### Details

Function details:

- `scorer_group_pct` returns the k-th top group percentage across districts. For example, if the group is Democratic voters and `k=3`, then the function returns the 3rd-highest fraction of Democratic voters across all districts. Can be used to target k VRA districts or partisan gerrymanders.

- `scorer_pop_dev` returns the maximum population deviation within a plan. Smaller values are closer to population parity, so use `maximize=FALSE` with this scorer.
- `scorer_splits` returns the fraction of counties that are split within a plan. Higher values have more county splits, so use `maximize=FALSE` with this scorer.
- `scorer_frac_kept` returns the fraction of edges kept in each district. Higher values mean more compactness.
- `scorer_polsby_popper` returns the  $m$ -th Polsby Popper score within a plan. Higher scores correspond to more compact districts. Use  $m=\text{ndists}/2$  to target the median compactness,  $m=1$  to target the minimum compactness.
- `scorer_status_quo` returns  $1 -$  the rescaled variation of information distance between the plan and the `existing_plan`. Larger values indicate the plan is closer to the existing plan.

### Value

A scoring function of class `redist_scorer` which returns a single numeric value per plan. Larger values are generally better for `frac_kept`, `group_pct`, and `polsby_popper` and smaller values are better for `splits` and `pop_dev`.

### Examples

```
data(iowa)
iowa_map <- redist_map(iowa, existing_plan = cd_2010, pop_tol = 0.05, total_pop = pop)

scorer_frac_kept(iowa_map)
scorer_status_quo(iowa_map)
scorer_group_pct(iowa_map, dem_08, tot_08, k = 2)
1.5*scorer_frac_kept(iowa_map) + 0.4*scorer_status_quo(iowa_map)
1.5*scorer_frac_kept(iowa_map) + scorer_frac_kept(iowa_map)*scorer_status_quo(iowa_map)
cbind(
  comp = scorer_frac_kept(iowa_map),
  sq = scorer_status_quo(iowa_map)
)
```

---

segregation_index	<i>Segregation index calculation for MCMC redistricting.</i>
-------------------	--

---

### Description

`redist.segcalc` calculates the dissimilarity index of segregation (see Massey & Denton 1987 for more details) for a specified subgroup under any redistricting plan.

**Usage**

```
segregation_index(
  map,
  group_pop,
  total_pop = map[[attr(map, "pop_col")]],
  .data = cur_plans()
)

redist.segcalc(plans, group_pop, total_pop)
```

**Arguments**

map	a <a href="#">redist_map</a> object
group_pop	A vector of populations for some subgroup of interest.
total_pop	A vector containing the populations of each geographic unit.
.data	a <a href="#">redist_plans</a> object
plans	A matrix of congressional district assignments or a redist object.

**Value**

`redist.segcalc` returns a vector where each entry is the dissimilarity index of segregation (Massey & Denton 1987) for each redistricting plan in `algot`.

**References**

Fifield, Benjamin, Michael Higgins, Kosuke Imai and Alexander Tarr. (2016) "A New Automated Redistricting Simulator Using Markov Chain Monte Carlo." Working Paper. Available at <http://imai.princeton.edu/research/files/redist.pdf>.

Massey, Douglas and Nancy Denton. (1987) "The Dimensions of Social Segregation". Social Forces.

**Examples**

```
data(fl25)
data(fl25_enum)
data(fl25_adj)

## Get an initial partition
init_plan <- fl25_enum$plans[, 5118]
fl25$init_plan <- init_plan

## 25 precinct, three districts - no pop constraint ##
fl_map <- redist_map(fl25, existing_plan = 'init_plan', adj = fl25_adj)
alg_253 <- redist_flip(fl_map, nsims = 10000)

## Get Republican Dissimilarity Index from simulations
# old: rep_dmi_253 <- redist.segcalc(alg_253, fl25$mccain, fl25$pop)
rep_dmi_253 <- seg_dissim(alg_253, fl25, mccain, pop) |>
```

```
redistmetrics::by_plan(ndists = 3)
```

---

subset_sampled	<i>Subset to sampled or reference draws</i>
----------------	---

---

### Description

Subset to sampled or reference draws

### Usage

```
subset_sampled(plans, matrix = TRUE)
```

```
subset_ref(plans, matrix = TRUE)
```

### Arguments

plans	the redist_plans object
matrix	if TRUE, the default, also subset the plans matrix. If the plans matrix is not needed, turning this off may save some time.

### Value

a redist\_plans object, with only rows corresponding to simulated (or reference) draws remaining.

---

summary.redist_plans	<i>Diagnostic information on sampled plans</i>
----------------------	--

---

### Description

Prints diagnostic information, which varies by algorithm. All algorithms compute the [plans\\_diversity\(\)](#) of the samples.

### Usage

```
## S3 method for class 'redist_plans'
summary(object, district = 1L, all_runs = TRUE, vi_max = 100, ...)
```

## Arguments

object	a <code>redist_plans</code> object
district	For R-hat values, which district to use for district-level summary statistics. We strongly recommend calling <code>match_numbers()</code> or <code>number_by()</code> before examining these district-level statistics.
all_runs	When there are multiple SMC runs, show detailed summary statistics for all runs (the default), or only the first run?
vi_max	The maximum number of plans to sample in computing the pairwise variation of information distance (sample diversity).
...	additional arguments (ignored)

## Details

For SMC and MCMC, if there are multiple runs/chains, R-hat values will be computed for each summary statistic. These values should be close to 1. If they are not, then there is too much between-chain variation, indicating that there are not enough samples. R-hat values are calculated after rank-normalization and folding. MCMC chains are split in half before R-hat is computed. For summary statistics that vary across districts, R-hat is calculated for the first district only.

For SMC, diagnostics statistics include:

- **Effective samples:** the effective sample size at each iteration, computed using the SMC weights. Larger is better. The percentage in parentheses is the ratio of the effective samples to the total samples.
- **Acceptance rate:** the fraction of drawn spanning trees which yield a valid redistricting plan within the population tolerance. Very small values (< 1%) can indicate a bottleneck and may lead to a lack of diversity.
- **Standard deviation of the log weights:** More variable weights (larger s.d.) indicate less efficient sampling. Values greater than 3 are likely problematic.
- **Maximum unique plans:** an upper bound on the number of unique redistricting plans that survive each stage. The percentage in parentheses is the ratio of this number to the total number of samples. Small values (< 100) indicate a bottleneck, which leads to a loss of sample diversity and a higher variance.
- **Estimated k parameter:** How many spanning tree edges were considered for cutting at each split. Mostly informational, though large jumps may indicate a need to increase `adapt_k_thresh`.
- **Bottleneck:** An asterisk will appear in the right column if a bottleneck appears likely, based on the values of the other statistics.

In the event of problematic diagnostics, the function will provide suggestions for improvement.

## Value

A data frame containing diagnostic information, invisibly.

**Examples**

```
data(iowa)
iowa_map <- redist_map(iowa, ndists = 4, pop_tol = 0.1)
plans <- redist_smc(iowa_map, 100)
summary(plans)
```

---

tally_var	<i>Tally a variable by district</i>
-----------	-------------------------------------

---

**Description**

Tally a variable by district

**Usage**

```
tally_var(map, x, .data = pl())
```

**Arguments**

map	a redist_map object
x	a variable to tally. Tidy-evaluated.
.data	a redist_plans object or matrix of plans

**Value**

a vector containing the tallied values by district and plan (column-major)

# Index

- \* **analysis**
  - get\_mh\_acceptance\_rate, 23
  - get\_sampling\_info, 25
- \* **analyze**
  - add\_reference, 5
  - classify\_plans, 6
  - compare\_plans, 6
  - competitiveness, 8
  - county\_splits, 13
  - distr\_compactness, 13
  - get\_plans\_matrix, 24
  - get\_plans\_weights, 24
  - group\_frac, 26
  - is\_county\_split, 28
  - last\_plan, 29
  - match\_numbers, 30
  - min\_move\_parity, 32
  - muni\_splits, 33
  - number\_by, 33
  - partisan\_metrics, 34
  - plan\_distances, 38
  - plans\_diversity, 37
  - plot.redist\_classified, 39
  - prec\_assignment, 42
  - prec\_cooccurrence, 43
  - proj, 45
  - pullback, 47
  - rbind.redist\_plans, 47
  - redist.district\_splits, 58
  - redist.multisplits, 69
  - redist.parity, 70
  - redist\_ci, 93
  - redist\_plans, 107
  - segregation\_index, 117
  - subset\_sampled, 119
  - summary.redist\_plans, 119
  - tally\_var, 121
- \* **data**
  - EPSG, 17
  - f125, 17
  - f1250, 18
  - f125\_adj, 19
  - f125\_enum, 20
  - f170, 20
  - iowa, 27
- \* **enumerate**
  - redist.calc.frontier.size, 48
  - redist.enumpart, 59
  - redist.init.enumpart, 63
  - redist.prep.enumpart, 83
  - redist.read.enumpart, 85
  - redist.run.enumpart, 88
- \* **plot**
  - plot.redist\_map, 41
  - plot.redist\_plans, 42
  - redist.diagplot, 55
  - redist.plot.adj, 70
  - redist.plot.contr\_pfd, 71
  - redist.plot.cores, 72
  - redist.plot.distr\_qtys, 73
  - redist.plot.hist, 75
  - redist.plot.majmin, 76
  - redist.plot.map, 76
  - redist.plot.plans, 78
  - redist.plot.scatter, 79
  - redist.plot.trace, 80
  - redist.plot.varinfo, 81
- \* **post**
  - redist.combine.mpi, 49
  - redist.ipw, 63
  - redist.smc\_is\_ci, 90
  - redist.uncoarsen, 92
- \* **prepare**
  - freeze, 21
  - get\_adj, 22
  - get\_existing, 23
  - get\_pop\_tol, 25
  - get\_target, 26

- is\_contiguous, 28
- make\_cores, 29
- merge\_by, 31
- plot.redist\_constr, 40
- plot.redist\_map, 41
- redist.coarsen.adjacency, 49
- redist.constraint.helper, 51
- redist.county.id, 52
- redist.county.relabel, 53
- redist.find.target, 60
- redist.findparams, 60
- redist.plot.penalty, 77
- redist.reduce.adjacency, 86
- redist.sink.plan, 89
- redist.subset, 91
- redist\_map, 101
- scorer-arith, 114
- scorer-combine, 115
- scorer\_group\_pct, 116
- \* simulate**
  - constraints, 9
  - redist.crsg, 53
  - redist.mcmc.mpi, 66
  - redist.rsg, 87
  - redist\_constr, 94
  - redist\_flip, 95
  - redist\_flip\_anneal, 99
  - redist\_mergesplit, 102
  - redist\_mergesplit\_parallel, 105
  - redist\_shortburst, 109
  - redist\_smc, 111
- \*.redist\_scorer (scorer-arith), 114
- +.redist\_scorer (scorer-arith), 114
- .redist\_scorer (scorer-arith), 114
- ?scorers, 110
- add\_constr\_compet (constraints), 9
- add\_constr\_compet(), 94
- add\_constr\_custom (constraints), 9
- add\_constr\_custom(), 94
- add\_constr\_edges\_rem (constraints), 9
- add\_constr\_edges\_rem(), 94
- add\_constr\_fry\_hold (constraints), 9
- add\_constr\_fry\_hold(), 94
- add\_constr\_grp\_hinge (constraints), 9
- add\_constr\_grp\_hinge(), 95
- add\_constr\_grp\_inv\_hinge (constraints), 9
- add\_constr\_grp\_inv\_hinge(), 95
- add\_constr\_grp\_pow (constraints), 9
- add\_constr\_grp\_pow(), 77, 78, 95
- add\_constr\_incumbency (constraints), 9
- add\_constr\_incumbency(), 95
- add\_constr\_log\_st (constraints), 9
- add\_constr\_log\_st(), 95
- add\_constr\_multisplits (constraints), 9
- add\_constr\_multisplits(), 95
- add\_constr\_polsby (constraints), 9
- add\_constr\_polsby(), 95
- add\_constr\_pop\_dev (constraints), 9
- add\_constr\_pop\_dev(), 95
- add\_constr\_segregation (constraints), 9
- add\_constr\_segregation(), 95
- add\_constr\_splits (constraints), 9
- add\_constr\_splits(), 95
- add\_constr\_status\_quo (constraints), 9
- add\_constr\_status\_quo(), 95
- add\_constr\_total\_splits (constraints), 9
- add\_constr\_total\_splits(), 95
- add\_reference, 5, 108
- as.matrix.redist\_plans, 108
- as.matrix.redist\_plans
  - (get\_plans\_matrix), 24
- as\_redist\_map (redist\_map), 101
- avg\_by\_prec, 5
- cbind(), 115
- cbind.redist\_scorer (scorer-combine), 115
- classify\_plans, 6
- classify\_plans(), 39
- combine\_scorers (scorer-combine), 115
- compare\_plans, 6
- compare\_plans(), 39
- competitiveness, 8
- constraints, 9, 112
- county\_splits, 13
- distr\_compactness, 13
- dplyr::summarize, 47
- EPSG, 17
- f125, 17, 20
- f1250, 18
- f125\_adj, 17, 19, 19
- f125\_enum, 17, 19, 20
- f170, 20

- freeze, 21
- geom\_boxplot, 74
- geom\_histogram, 75
- geom\_line, 80
- geom\_point, 79
- get\_adj, 22, 102
- get\_existing, 23
- get\_mh\_acceptance\_rate, 23
- get\_plans\_matrix, 24, 108
- get\_plans\_weights, 24, 108
- get\_pop\_tol, 25
- get\_sampling\_info, 25, 108
- get\_target, 26
- group\_frac, 26
- hclust(), 6
- hist.redist\_plans (redist.plot.hist), 75
- iowa, 27
- is\_contiguous, 28
- is\_county\_split, 28, 108
- last\_plan, 29
- loo, 113
- make\_cores, 29
- makeCluster(), 106
- match\_numbers, 30, 108
- merge\_by, 31, 47, 102
- min\_move\_parity, 32
- muni\_splits, 33
- number\_by, 33, 108
- parallel::makeCluster(), 97
- partisan\_metrics, 34
- pl, 36
- plan\_distances, 38, 108
- plan\_distances(), 6
- plan\_parity (redist.parity), 70
- plans\_diversity, 37
- plans\_diversity(), 119
- plot.redist\_classified, 39
- plot.redist\_classified(), 6
- plot.redist\_constr, 40
- plot.redist\_constr(), 11
- plot.redist\_map, 41, 102
- plot.redist\_plans, 42, 108
- prec\_assignment, 42, 108
- prec\_cooccurrence, 43
- print.redist\_classified, 43
- print.redist\_constr, 44
- print.redist\_map, 44
- print.redist\_plans, 45
- proj, 45
- proj\_avg (proj), 45
- proj\_avg(), 5
- proj\_constr (proj), 45
- proj\_constr(), 72
- proj\_distr (proj), 45
- pullback, 47, 108
- rbind.redist\_plans, 47
- redist.adjacency, 48
- redist.calc.frontier.size, 48
- redist.coarsen.adjacency, 49
- redist.combine.mpi, 49
- redist.compactness (distr\_compactness), 13
- redist.competitiveness (competitiveness), 8
- redist.constraint.helper, 51
- redist.county.id, 52
- redist.county.relabel, 53
- redist.crsg, 53
- redist.diagplot, 55
- redist.dist.pop.overlap, 57
- redist.distances (plan\_distances), 38
- redist.district\_splits, 58
- redist.enumpart, 59
- redist.find.target, 60
- redist.findparams, 60
- redist.freeze (freeze), 21
- redist.group.percent (group\_frac), 26
- redist.identify.cores (make\_cores), 29
- redist.init.enumpart, 63
- redist.ipw, 63
- redist.mcmc.mpi, 66
- redist.metrics (partisan\_metrics), 34
- redist.multisplits, 69
- redist.muni\_splits (muni\_splits), 33
- redist.parity, 70
- redist.plot.adj, 41, 70
- redist.plot.contr\_pfd, 71
- redist.plot.contr\_pfd(), 45
- redist.plot.cores, 72
- redist.plot.cores(), 30
- redist.plot.distr\_qtys, 73

- redist.plot.hist, 75
- redist.plot.majmin, 76
- redist.plot.map, 41, 76
- redist.plot.penalty, 77
- redist.plot.plans, 42, 78
- redist.plot.scatter, 79
- redist.plot.trace, 80
- redist.plot.varinfo, 81
- redist.plot.wted.adj, 81
- redist.prec.pop.overlap, 82
- redist.prep.enumpart, 83
- redist.random.subgraph, 84
- redist.read.enumpart, 85
- redist.reduce.adjacency, 86
- redist.reorder, 86
- redist.rsg, 87
- redist.run.enumpart, 88
- redist.segcalc (segregation\_index), 117
- redist.sink.plan, 89
- redist.smc\_is\_ci, 90
- redist.splits (county\_splits), 13
- redist.subset, 91
- redist.uncoarsen, 92
- redist.wted.adj, 92
- redist\_ci, 93
- redist\_constr, 40, 94
- redist\_constr(), 9, 10, 112
- redist\_flip, 95
- redist\_flip(), 109
- redist\_flip\_anneal, 99
- redist\_map, 8, 11, 13, 14, 22, 25, 26, 29, 31–34, 39, 61, 70, 72, 96, 100, 101, 103, 105, 107, 110, 116, 118
- redist\_map(), 94, 112
- redist\_mcmc\_ci (redist\_ci), 93
- redist\_mergesplit, 102, 110
- redist\_mergesplit(), 9, 94, 105, 109
- redist\_mergesplit\_parallel, 105
- redist\_plans, 7, 8, 13, 14, 26, 29, 33, 34, 37, 39, 42, 43, 45–47, 70, 75, 80, 93, 99, 104, 106, 107, 114, 118, 120
- redist\_quantile\_trunc, 109
- redist\_quantile\_trunc(), 114
- redist\_shortburst, 109
- redist\_smc, 103, 111
- redist\_smc(), 9, 94, 110
- redist\_smc\_ci (redist\_ci), 93
- redistmetrics::prep\_perims(), 14, 116
- scorer-arith, 114, 115
- scorer-combine, 115
- scorer\_frac\_kept (scorer\_group\_pct), 116
- scorer\_group\_pct, 116
- scorer\_multisplits (scorer\_group\_pct), 116
- scorer\_polsby\_popper (scorer\_group\_pct), 116
- scorer\_pop\_dev (scorer\_group\_pct), 116
- scorer\_splits (scorer\_group\_pct), 116
- scorer\_status\_quo (scorer\_group\_pct), 116
- scorers, 115
- scorers (scorer\_group\_pct), 116
- segregation\_index, 117
- set\_adj (get\_adj), 22
- set\_pop\_tol (get\_pop\_tol), 25
- sf::st\_transform(), 17
- str(), 94
- subset\_ref, 108
- subset\_ref (subset\_sampled), 119
- subset\_sampled, 108, 119
- summarize(), 93
- summary.redist\_plans, 108, 119
- tally\_var, 121
- tibble::tibble, 101, 102
- weights.redist\_plans (get\_plans\_weights), 24