Package 'chopin'

April 3, 2026

Title Spatial Parallel Computing by Hierarchical Data Partitioning

Version 0.9.9-5

Description Geospatial data computation is parallelized by grid, hierarchy, or raster files. Based on 'future' (Bengtsson, 2024 <doi:10.32614/CRAN.package.future>) and 'mirai' (Gao et al., 2025 <doi:10.32614/CRAN.package.mirai>) parallel back-ends, 'terra' (Hijmans et al., 2025 <doi:10.32614/CRAN.package.terra>) and 'sf' (Pebesma et al., 2024 <doi:10.32614/CRAN.package.sf>) functions as well as convenience functions in the package can be distributed over multiple threads. The simplest way of parallelizing generic geospatial computation is to start from par_pad_*() functions to par_grid(), par_hierarchy(), or par_multirasters() functions. Virtually any functions accepting classes in 'terra' or 'sf' packages can be used in the three parallelization functions. A common raster-vector overlay operation is provided as a function extract_at(), which uses 'exactextractr' (Baston, 2023 <doi:10.32614/CRAN.package.exactextractr>), with options for kernel weights for summarizing raster values at vector geometries. Other convenience functions for vector-vector operations including simple areal interpolation (summarize_aw()) and summation of exponentially decaying weights (summarize_sedc()) are also provided.

License MIT + file LICENSE

URL https://docs.ropensci.org/chopin/, https://github.com/ropensci/chopin

BugReports https://github.com/ropensci/chopin/issues

Depends R (>= 4.1)

SystemRequirements netcdf

Encoding UTF-8

LazyData true

LazyDataCompression xz

RoxygenNote 7.3.3

Imports anticlust, cli, dplyr (>= 1.1.0), exactextractr (>= 0.8.2), future, future.apply, igraph, methods, rlang, sf (>= 1.0-10), sfheaders, stars (>= 0.6-0), terra (>= 1.7-18), mirai (>= 1.3.0), collapse

Suggests covr, devtools, targets, DiagrammeR, future.mirai, knitr, lifecycle, rmarkdown, spatstat.random, testthat (>= 3.0.0), units, withr, dggridR, h3r

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation no

Author Insang Song [aut, cre] (ORCID: <https://orcid.org/0000-0001-8732-3256>), Kyle Messier [aut, ctb] (ORCID: <https://orcid.org/0000-0001-9508-9623>), Alec L. Robitaille [rev] (Alec reviewed the package version 0.6.3 for rOpenSci, see <https://github.com/ropensci/software-review/issues/638>), Eric R. Scott [rev] (Eric reviewed the package version 0.6.3 for rOpenSci, see <https://github.com/ropensci/software-review/issues/638>)

Maintainer Insang Song <geoissong@gmail.com>

Repository CRAN

Date/Publication 2026-04-03 12:40:02 UTC

Contents

extract_at *Extract raster values with point buffers or polygons*

Description

Extract raster values with point buffers or polygons

Usage

```
extract_at(x, y, ...)

## S4 method for signature 'SpatRaster,sf'
extract_at(
  x = NULL,
  y = NULL,
  id = NULL,
  func = "mean",
  extent = NULL,
  radius = NULL,
  out_class = "sf",
  kernel = NULL,
  kernel_func = stats::weighted.mean,
  bandwidth = NULL,
  max_cells = 3e+07,
  .standalone = TRUE,
  ...
)

## S4 method for signature 'character,character'
extract_at(
  x = NULL,
  y = NULL,
  id = NULL,
  func = "mean",
  extent = NULL,
  radius = NULL,
  out_class = "sf",
  kernel = NULL,
  kernel_func = stats::weighted.mean,
  bandwidth = NULL,
  max_cells = 3e+07,
  .standalone = TRUE,
  ...
)

## S4 method for signature 'SpatRaster,character'
extract_at(
```

```
  x = NULL,
  y = NULL,
  id = NULL,
  func = "mean",
  extent = NULL,
  radius = NULL,
  out_class = "sf",
  kernel = NULL,
  kernel_func = stats::weighted.mean,
  bandwidth = NULL,
  max_cells = 3e+07,
  .standalone = TRUE,
  ...
)

## S4 method for signature 'SpatRaster,SpatVector'
extract_at(
  x = NULL,
  y = NULL,
  id = NULL,
  func = "mean",
  extent = NULL,
  radius = NULL,
  out_class = "sf",
  kernel = NULL,
  kernel_func = stats::weighted.mean,
  bandwidth = NULL,
  max_cells = 3e+07,
  .standalone = TRUE,
  ...
)

## S4 method for signature 'character,sf'
extract_at(
  x = NULL,
  y = NULL,
  id = NULL,
  func = "mean",
  extent = NULL,
  radius = NULL,
  out_class = "sf",
  kernel = NULL,
  kernel_func = stats::weighted.mean,
  bandwidth = NULL,
  max_cells = 3e+07,
  .standalone = TRUE,
  ...
)
```

```
## S4 method for signature 'character,SpatVector'
extract_at(
  x = NULL,
  y = NULL,
  id = NULL,
  func = "mean",
  extent = NULL,
  radius = NULL,
  out_class = "sf",
  kernel = NULL,
  kernel_func = stats::weighted.mean,
  bandwidth = NULL,
  max_cells = 3e+07,
  .standalone = TRUE,
  ...
)
```

Arguments

x	SpatRaster object or file path(s) with extensions that are GDAL-compatible. When multiple file paths are used, the rasters must have the same extent and resolution.
y	sf/SpatVector object or file path.
...	Placeholder.
id	character(1). Unique identifier of each point.
func	function taking one numeric vector argument. Default is "mean" for all supported signatures in arguments x and y.
extent	numeric(4) or SpatExtent. Extent of clipping vector. It only works with points of character(1) file path.
radius	numeric(1). Buffer radius.
out_class	character(1). Output class. One of sf or terra.
kernel	character(1). Name of a kernel function One of "uniform", "triweight", "quartic", and "epanechnikov"
kernel_func	function. Kernel function to apply to the extracted values. Default is stats::weighted.mean()
bandwidth	numeric(1). Kernel bandwidth.
max_cells	integer(1). Maximum number of cells in memory.
.standalone	logical(1). Default is TRUE, which means that the function will be executed in a standalone mode. When using this function in par_* functions, set this to FALSE.

Details

Inputs are preprocessed in different ways depending on the class.

- Vector inputs in y: sf is preferred, thus character and SpatVector inputs will be converted to sf object. If radius is not NULL, sf::st_buffer is used to generate circular buffers as subsequent raster-vector overlay is done with exactextractr::exact_extract.
- Raster input in x: SpatRaster is preferred. If the input is not SpatRaster, it will be converted to SpatRaster object.

Value

A data.frame object with summarized raster values with respect to the mode (polygon or buffer) and the function.

Author(s)

Insang Song <geoissong@gmail.com>

See Also

Other Macros for calculation: kernelfunction(), summarize_aw(), summarize_pp(), summarize_sedc(), summarize_st()

Examples

```
ncpath <- system.file("gpkg/nc.gpkg", package = "sf")
rastpath <- file.path(tempdir(), "test.tif")

nc <- terra::vect(ncpath)
nc <- terra::project(nc, "EPSG:5070")
rrast <- terra::rast(nc, nrow = 300, ncol = 660)
terra::values(rrast) <- rgamma(1.98e5, 4, 2)
rpnt <- terra::spatSample(rrast, 16L, as.points = TRUE)
rpnt$pid <- sprintf("ID-%02d", seq(1, 16))

extract_at(rrast, rpnt, "pid", "mean", radius = 1000)
extract_at(rrast, nc, "NAME", "mean")
extract_at(rrast, ncpath, "NAME", "mean")
# Using SpatRaster object
suppressWarnings(
  extract_at(
    rrast, ncpath, "NAME", "mean",
    kernel = "epanechnikov",
    bandwidth = 1e5
  )
)
# Using raster path
terra::writeRaster(rrast, rastpath, overwrite = TRUE)
suppressWarnings(
  extract_at(
    rastpath, ncpath, "NAME", "mean",
    kernel = "epanechnikov",
    bandwidth = 1e5
  )
)
```

ncpoints *Mildly clustered points in North Carolina, United States*

Description

Mildly clustered points in North Carolina, United States

Usage

```
ncpoints
```

Format

A data frame with 2,304 rows and two variables:

X X coordinate

Y Y coordinate

Note

Coordinates are in EPSG:5070 (Conus Albers Equal Area)

Source

sf package data nc

See Also

Other Dataset: prediction_grid

Examples

```
data("ncpoints", package = "chopin")
```

par_convert_f *Map specified arguments to others in literals*

Description

This function creates a new function that wraps an existing function, remapping the argument names based on a user-specified literal mapping. Specifically, arguments passed to the new function with names on the left-hand side of the mapping are renamed to the corresponding right-hand side names before being passed to the original function. Users map two arguments without x and/or y to standardize the argument names, x and y, to the target function. This function is particularly useful to parallelize functions for spatial data outside sf and terra packages that do not have arguments named x and/or y. par_* functions could detect such functions by wrapping nonstandardized functions to parallelize the computation.

Usage

```
par_convert_f(fun, ...)
```

Arguments

fun	A function to be wrapped.
...	A set of named arguments representing the mapping from the new function's argument names (left-hand side) to the original function's argument names (right-hand side). For example, `x = group, y = score` maps argument `x` to `group` and `y` to `score`.

Value

A new function that accepts the remapped arguments and calls the original function.

Examples

```
# Define an original function that expects arguments 'group' and 'score'
original_fun <- function(group, score, home = FALSE) {
  list(group = group, score = score, home = home)
}

# Create a new function that maps 'x' to 'group' and 'y' to 'score'
new_fun <- par_convert_f(original_fun, x = group, y = score)

# Call the new function using the new argument names
result <- new_fun(x = 10, y = 20)
print(result)
```

---

par_grid *Parallelize spatial computation over the computational grids*

Description

future::multicore, future::multisession, future::cluster future.mirai::mirai_multisession in future::plan will parallelize the work in each grid. For details of the terminology in `future` package, refer to `future::plan`. This function assumes that users have one raster file and a sizable and spatially distributed target locations. Each thread will process the nearest integer of $|N_g| / |N_t|$ grids where $|N_g|$ denotes the number of grids and $|N_t|$ denotes the number of threads.

Usage

```
par_grid(grids, fun_dist, ..., pad_y = FALSE, .debug = FALSE)
```

Arguments

grids	List of two sf/SpatVector objects. Computational grids. It takes a strict assumption that the grid input is an output of par_pad_grid.
fun_dist	sf, terra or chopin functions. This function should have x and y arguments.
...	Arguments passed to the argument fun_dist.
pad_y	logical(1). Whether to filter y with the padded grid. Should be TRUE when x is where the values are calculated. Default is FALSE. In the reverse case, like terra::extent or exactextractr::exact_extract, the raster (x) extent should be set with the padded grid.
.debug	logical(1). Default is FALSE. Otherwise, if a unit computation fails, the error message and the CGRIDID value where the error occurred will be included in the output.

Value

a data.frame object with computation results. For entries of the results, consult the documentation of the function put in fun_dist argument.

Note

In dynamic dots (...), fun_dist arguments should include x and y where sf/terra class objects or file paths are accepted. Virtually any sf/terra functions that accept two arguments can be put in fun_dist; however, be advised that some spatial operations do not necessarily give the exact result from what would have been done with one thread. For example, distance calculated through this function may return the lower value than actual because the computational region was reduced. This would be the case especially where the target features are spatially sparsely distributed.

Author(s)

Insang Song <geoissong@gmail.com>

See Also

future::multisession, future::multicore, future::cluster, future.mirai::mirai_multisession, future::plan, par_convert_f

Other Parallelization: par_cut_coords(), par_grid_mirai(), par_hierarchy(), par_hierarchy_mirai(), par_make_dggrid(), par_make_grid(), par_make_h3(), par_merge_grid(), par_multirasters(), par_multirasters_mirai(), par_pad_balanced(), par_pad_grid(), par_split_list()

Examples

```
lastpar <- par(mfrow = c(1, 1))
library(sf)
library(future)
library(future.mirai)
options(sf_use_s2 = FALSE)
plan(mirai_multisession, workers = 2)
ncpath <- system.file("shape/nc.shp", package = "sf")
```

```
ncpoly <- sf::st_read(ncpath)
ncpoly <- sf::st_transform(ncpoly, "EPSG:5070")

# sf object
ncpnts <- sf::st_sample(ncpoly, 2000)
ncpnts <- sf::st_as_sf(ncpnts)
ncpnts$pid <- seq_len(nrow(ncpnts))

# file path
rrast <- terra::rast(ncpoly, nrow = 600, ncol = 1320)
terra::values(rrast) <- rgamma(7.92e5, 4, 2)

# Using raster path
rastpath <- file.path(tempdir(), "ncelev.tif")
terra::writeRaster(rrast, rastpath, overwrite = TRUE)

nccompreg <-
  chopin::par_pad_grid(
    input = ncpnts,
    mode = "grid",
    nx = 4L,
    ny = 2L,
    padding = 5e3L
  )
res <-
  par_grid(
    grids = nccompreg,
    fun_dist = extract_at,
    x = rastpath,
    y = ncpnts,
    qsegs = 90L,
    radius = 5e3L,
    id = "pid"
  )
future::plan(future::sequential)
mirai::daemons(0)
par(lastpar)
```

`par_grid_mirai` *Parallelize spatial computation over the computational grids*

Description

mirai::daemons will set the parallel backend then mirai::mirai_map will parallelize the work in each grid. For details of the terminology in `mirai` package, refer to `mirai::mirai`. This function assumes that users have one raster file and a sizable and spatially distributed target locations. Each thread will process the nearest integer of $|N_g| / |N_t|$ grids where $|N_g|$ denotes the number of grids and $|N_t|$ denotes the number of threads.

Usage

```
par_grid_mirai(grids, fun_dist, ..., pad_y = FALSE, .debug = TRUE)
```

Arguments

grids	List of two sf/SpatVector objects. Computational grids. It takes a strict assumption that the grid input is an output of par_pad_grid.
fun_dist	sf, terra or chopin functions. This function should have x and y arguments.
...	Arguments passed to the argument fun_dist.
pad_y	logical(1). Whether to filter y with the padded grid. Should be TRUE when x is where the values are calculated. Default is FALSE. In the reverse case, like terra::extent or exactextractr::exact_extract, the raster (x) extent should be set with the padded grid.
.debug	logical(1). Default is FALSE. Otherwise, if a unit computation fails, the error message and the CGRIDID value where the error occurred will be included in the output.

Value

a data.frame object with computation results. For entries of the results, consult the documentation of the function put in fun_dist argument.

Note

In dynamic dots (...), fun_dist arguments should include x and y where sf/terra class objects or file paths are accepted. Virtually any sf/terra functions that accept two arguments can be put in fun_dist; however be advised that some spatial operations do not necessarily give the exact result from what would have been done with one thread. For example, distance calculated through this function may return the lower value than actual because the computational region was reduced. This would be the case especially where the target features are spatially sparsely distributed.

Author(s)

Insang Song <geoissong@gmail.com>

See Also

mirai::daemons, mirai::mirai_map, par_convert_f

Other Parallelization: par_cut_coords(), par_grid(), par_hierarchy(), par_hierarchy_mirai(), par_make_dggrid(), par_make_grid(), par_make_h3(), par_merge_grid(), par_multirasters(), par_multirasters_mirai(), par_pad_balanced(), par_pad_grid(), par_split_list()

Examples

```
lastpar <- par(mfrow = c(1, 1))
library(sf)
library(mirai)
options(sf_use_s2 = FALSE)
```

```
daemons(4)
ncpath <- system.file("shape/nc.shp", package = "sf")
ncpoly <- sf::st_read(ncpath)
ncpoly <- sf::st_transform(ncpoly, "EPSG:5070")

# sf object
ncpnts <-
  sf::st_sample(ncpoly, 2000)
ncpnts <- sf::st_as_sf(ncpnts)
ncpnts$pid <- seq_len(nrow(ncpnts))

# file path
rrast <- terra::rast(ncpoly, nrow = 600, ncol = 1320)
terra::values(rrast) <- rgamma(7.92e5, 4, 2)
# Using raster path
rastpath <- file.path(tempdir(), "ncelev.tif")
terra::writeRaster(rrast, rastpath, overwrite = TRUE)

nccompreg <-
  chopin::par_pad_grid(
    input = ncpnts,
    mode = "grid",
    nx = 4L,
    ny = 2L,
    padding = 5e3L
  )
res <-
  par_grid_mirai(
    grids = nccompreg,
    fun_dist = extract_at,
    x = rastpath,
    y = ncpnts,
    qsegs = 90L,
    radius = 5e3L,
    id = "pid"
  )
mirai::daemons(0L)
par(lastpar)
```

`par_hierarchy` *Parallelize spatial computation by hierarchy in input data*

Description

"Hierarchy" refers to a system, which divides the entire study region into multiple subregions. It is oftentimes reflected in an area code system (e.g., FIPS for US Census geographies and Nomenclature of Territorial Units for Statistics (NUTS), etc.). `future::multisession`, `future::multicore`, `future::cluster`, `future.mirai::mirai_multisession` in `future::plan` will parallelize the work by splitting lower level features into several higher level feature group. For details of the

terminology in future package, please refer to future::plan documentation. Each thread will process the number of lower level features in each higher level feature. Be advised that accessing the same file simultaneously with multiple processes may result in errors.

Usage

```
par_hierarchy(
  regions,
  regions_id = NULL,
  length_left = NULL,
  pad = 0,
  pad_y = FALSE,
  fun_dist,
  ...,
  .debug = FALSE
)
```

Arguments

regions	sf/SpatVector object. Computational regions. Only polygons are accepted.
regions_id	character(1). Name of unique ID field in regions. The regions will be split by the common level value.
length_left	integer(1). Length of the first characters of the regions_id values. Default is NULL, which will not manipulate the regions_id values. If the number of characters is not consistent (for example, numerics), the function will alert the user.
pad	numeric(1). Padding distance for each subregion defined by regions_id or trimmed regions_id values. in linear unit of coordinate system. Default is 0, which means each subregion is used as is. If the value is greater than 0, the subregion will be buffered by the value. The padding distance will be applied to x (pad_y = FALSE) or y (pad_y = TRUE) to filter the data.
pad_y	logical(1). Whether to filter y with the padded grid. Should be TRUE when x is where the values are calculated. Default is FALSE. In the reverse case, like terra::extent or exactextractr::exact_extract, the raster (x) should be scoped with the padded grid.
fun_dist	sf, terra, or chopin functions. This function should have x and y arguments.
...	Arguments passed to the argument fun_dist.
.debug	logical(1). Default is FALSE If a unit computation fails, the error message and the regions_id value where the error occurred will be included in the output.

Details

In dynamic dots (...), fun_dist arguments should include x and y where sf/terra class objects or file paths are accepted. Hierarchy is interpreted by the regions_id argument first. regions_id is assumed to be a field name in the x or y argument object. It is expected that regions represents the higher level boundaries and x or y in fun_dist is the lower level boundaries. However, if that is not the case, with trim argument, the function will generate the higher level codes from regions_id by

extracting left-t Whether x or y is searched is determined by pad_y value. pad_y = TRUE will make the function attempt to find regions_id in x, whereas pad_y = FALSE will look for regions_id at y. If the regions_id doesn't exist in x or y, the function will utilize spatial relationship (intersects) to filter the data. Note that dispatching computation by subregions based on the spatial relationship may lead to a slight discrepancy in the result. For example, if the higher and lower level features are not perfectly aligned, there may be some features that are not included or duplicated in the subregions. The function will alert the user if spatial relation- ship is used to filter the data.

Value

a data.frame object with computation results. For entries of the results, consult the function used in fun_dist argument.

Note

Virtually any sf/terra functions that accept two arguments can be put in fun_dist; however, be advised that some spatial operations do not necessarily give the exact result from what would have been done with single thread. For example, distance calculated through this function may return the lower value than actual because the computational region was reduced. This would be the case especially where the target features are spatially sparsely distributed.

Author(s)

Insang Song <geoissong@gmail.com>

See Also

future::multisession, future::multicore, future::cluster, future.mirai::mirai_multisession, future::plan, par_convert_f

Other Parallelization: par_cut_coords(), par_grid(), par_grid_mirai(), par_hierarchy_mirai(), par_make_dggrid(), par_make_grid(), par_make_h3(), par_merge_grid(), par_multirasters(), par_multirasters_mirai(), par_pad_balanced(), par_pad_grid(), par_split_list()

Examples

```
lastpar <- par(mfrow = c(1, 1))
library(terra)
library(sf)
library(future)
library(future.mirai)
options(sf_use_s2 = FALSE)
future::plan(future.mirai::mirai_multisession, workers = 2)

nccnty <- sf::st_read(
  system.file("shape/nc.shp", package = "sf")
)
nccnty <- sf::st_transform(nccnty, "EPSG:5070")
nccnty <- nccnty[seq_len(30L), ]

nccntygrid <- sf::st_make_grid(nccnty, n = c(200, 100))
nccntygrid <- sf::st_as_sf(nccntygrid)
```

```
nccntygrid$GEOID <- sprintf("%05d", seq_len(nrow(nccntygrid)))
nccntygrid <- sf::st_intersection(nccntygrid, nccnty)

rrast <- terra::rast(nccnty, nrow = 600, ncol = 1320)
terra::values(rrast) <- rgamma(7.92e5, 4, 2)

# Using raster path
rastpath <- file.path(tempdir(), "ncelev.tif")
terra::writeRaster(rrast, rastpath, overwrite = TRUE)

ncsamp <-
  sf::st_sample(
    nccnty,
    size = 1e4L
  )
# sfc to sf
ncsamp <- sf::st_as_sf(ncsamp)
# assign ID
ncsamp$kid <- sprintf("K-%05d", seq_len(nrow(ncsamp)))
res <-
  par_hierarchy(
    regions = nccnty,
    regions_id = "FIPS",
    fun_dist = extract_at,
    y = nccntygrid,
    x = rastpath,
    id = "GEOID",
    func = "mean"
  )
future::plan(future::sequential)
mirai::daemons(0)
par(lastpar)
```

`par_hierarchy_mirai` *Parallelize spatial computation by hierarchy in input data*

Description

"Hierarchy" refers to a system, which divides the entire study region into multiple subregions. It is usually reflected in an area code system (e.g., FIPS for US Census geographies and Nomenclature of Territorial Units for Statistics (NUTS), etc.). mirai::daemons will set the parallel backend then mirai::mirai_map will the work by splitting lower level features into several higher level feature group. For details of the terminology in `mirai` package, refer to `mirai::mirai`. Each thread will process the number of lower level features in each higher level feature. Be advised that accessing the same file simultaneously with multiple processes may result in errors.

Usage

```
par_hierarchy_mirai(
```

```
  regions,
  regions_id = NULL,
  length_left = NULL,
  pad = 0,
  pad_y = FALSE,
  fun_dist,
  ...,
  .debug = TRUE
)
```

Arguments

regions	sf/SpatVector object. Computational regions. Only polygons are accepted.
regions_id	character(1). Name of unique ID field in regions. The regions will be split by the common level value.
length_left	integer(1). Length of the first characters of the regions_id values. Default is NULL, which will not manipulate the regions_id values. If the number of characters is not consistent (for example, numerics), the function will alert the user.
pad	numeric(1). Padding distance for each subregion defined by regions_id or trimmed regions_id values. in linear unit of coordinate system. Default is 0, which means each subregion is used as is. If the value is greater than 0, the subregion will be buffered by the value. The padding distance will be applied to x (pad_y = FALSE) or y (pad_y = TRUE) to filter the data.
pad_y	logical(1). Whether to filter y with the padded grid. Should be TRUE when x is where the values are calculated. Default is FALSE. In the reverse case, like terra::extent or exactextractr::exact_extract, the raster (x) should be scoped with the padded grid.
fun_dist	sf, terra, or chopin functions. This function should have x and y arguments.
...	Arguments passed to the argument fun_dist.
.debug	logical(1). Default is FALSE If a unit computation fails, the error message and the regions_id value where the error occurred will be included in the output.

Details

In dynamic dots (...), fun_dist arguments should include x and y where sf/terra class objects or file paths are accepted. Hierarchy is interpreted by the regions_id argument first. regions_id is assumed to be a field name in the x or y argument object. It is expected that regions represents the higher level boundaries and x or y in fun_dist is the lower level boundaries. However, if that is not the case, with trim argument, the function will generate the higher level codes from regions_id by extracting the code from the left end (controlled by length_left). Whether x or y is searched is determined by pad_y value. pad_y = TRUE will make the function attempt to find regions_id in x, whereas pad_y = FALSE will look for regions_id at y. If the regions_id doesn't exist in x or y, the function will utilize spatial relationship (intersects) to filter the data. Note that dispatching computation by subregions based on the spatial relationship may lead to a slight discrepancy in the result. For example, if the higher and lower level features are not perfectly aligned, there may be some features that are not included or duplicated in the subregions. The function will alert the user if spatial relation- ship is used to filter the data.

Value

a data.frame object with computation results. For entries of the results, consult the function used in `fun_dist` argument.

Note

Virtually any sf/terra functions that accept two arguments can be put in `fun_dist`; however, be advised that some spatial operations do not necessarily give the exact result from what would have been done with one thread. For example, distance calculated through this function may return the lower value than actual because the computational region was reduced. This would be the case especially where the target features are spatially sparsely distributed.

Author(s)

Insang Song <geoissong@gmail.com>

See Also

`mirai::mirai_map`, `mirai::daemons`, `par_convert_f`

Other Parallelization: `par_cut_coords()`, `par_grid()`, `par_grid_mirai()`, `par_hierarchy()`, `par_make_dggrid()`, `par_make_grid()`, `par_make_h3()`, `par_merge_grid()`, `par_multirasters()`, `par_multirasters_mirai()`, `par_pad_balanced()`, `par_pad_grid()`, `par_split_list()`

Examples

```
lastpar <- par(mfrow = c(1, 1))
library(terra)
library(sf)
library(mirai)
options(sf_use_s2 = FALSE)
mirai::daemons(4)

nccnty <- sf::st_read(
  system.file("shape/nc.shp", package = "sf")
)
nccnty <- sf::st_transform(nccnty, "EPSG:5070")

nccntygrid <- sf::st_make_grid(nccnty, n = c(200, 100))
nccntygrid <- sf::st_as_sf(nccntygrid)
nccntygrid$GEOID <- sprintf("%05d", seq_len(nrow(nccntygrid)))
nccntygrid <- sf::st_intersection(nccntygrid, nccnty)

rrast <- terra::rast(nccnty, nrow = 600, ncol = 1320)
terra::values(rrast) <- rgamma(7.92e5, 4, 2)

# Using raster path
rastpath <- file.path(tempdir(), "ncelev.tif")
terra::writeRaster(rrast, rastpath, overwrite = TRUE)

ncsamp <-
  sf::st_sample(
```

```
    nccnty,
    size = 1e4L
  )
# sfc to sf
ncsamp <- sf::st_as_sf(ncsamp)
# assign ID
ncsamp$kid <- sprintf("K-%05d", seq_len(nrow(ncsamp)))
res <-
  par_hierarchy_mirai(
    regions = nccnty,
    regions_id = "FIPS",
    fun_dist = extract_at,
    y = nccntygrid,
    x = rastpath,
    id = "GEOID",
    func = "mean",
    .debug = TRUE
  )
mirai::daemons(0L)
par(lastpar)
```

`par_make_dggrid` *Convert DGGRID indices to sf object*

Description

This function converts DGGRID indices to an `sf` object. It requires the `dggridR` package to be installed.

Usage

```
par_make_dggrid(x, res = 8L, topology = "HEXAGON")
```

Arguments

`x`	sf object.
`res`	integer(1). DGGRID resolution. Default is 8L.
`topology`	character(1). Topology type, either "HEXAGON" or "SQUARE". Default is "HEXAGON".

Details

`dggridR::dgconstruct` is used to create a DGGRID object with the specified resolution. All arguments in this function are used as default values other than `res` and `topology`.

Value

An `sf` object with polygons representing the DGGRID indices.

Author(s)

Insang Song

See Also

Other Parallelization: par_cut_coords(), par_grid(), par_grid_mirai(), par_hierarchy(), par_hierarchy_mirai(), par_make_grid(), par_make_h3(), par_merge_grid(), par_multirasters(), par_multirasters_mirai(), par_pad_balanced(), par_pad_grid(), par_split_list()

Examples

```
lastpar <- par(mfrow = c(1, 1))
library(sf)
if (rlang::is_installed("dggridR")) {
  library(dggridR)
  options(sf_use_s2 = FALSE)
  ncpath <- system.file("shape/nc.shp", package = "sf")
  nc <- read_sf(ncpath)
  nc <- st_transform(nc, "EPSG:4326")
  nc_comp_region_dggrid <-
    par_make_dggrid(
      nc,
      res = 8L,
      topology = "HEXAGON"
    )
  plot(sf::st_geometry(nc_comp_region_dggrid))
}
par(lastpar)
```

---

par_make_h3 *Convert a input sf object to H3 hexagons*

Description

This function converts an input `sf` to an `sf` object with H3 hexagons. It requires the `h3r` package to be installed.

Usage

```
par_make_h3(x, res = 5L)
```

Arguments

`x`	sf object.
`res`	integer(1). H3 resolution. Default is 5L.

Details

Non-polygon `x` will be converted to polygons using `sf::st_concave_hull`. If the input is not convertible to polygons, the function will throw an error.

Value

An `sf` object with polygons representing the H3 indices.

Author(s)

Insang Song

See Also

Other Parallelization: `par_cut_coords()`, `par_grid()`, `par_grid_mirai()`, `par_hierarchy()`, `par_hierarchy_mirai()`, `par_make_dggrid()`, `par_make_grid()`, `par_merge_grid()`, `par_multirasters()`, `par_multirasters_mirai()`, `par_pad_balanced()`, `par_pad_grid()`, `par_split_list()`

Examples

```
lastpar <- par(mfrow = c(1, 1))
library(sf)
if (rlang::is_installed("h3r")) {
library(h3r)
options(sf_use_s2 = FALSE)
ncpath <- system.file("shape/nc.shp", package = "sf")
nc <- read_sf(ncpath)
nc <- st_transform(nc, "EPSG:4326")
# note that it will throw a warning if
# the input is MULTIPOLYGON.
nc_comp_region_h3 <-
  suppressWarnings(
    par_make_h3(
      nc,
      res = 5L
    )
  )
plot(sf::st_geometry(nc_comp_region_h3))
}
par(lastpar)
```

`par_merge_grid` *Merge adjacent grid polygons with given rules*

Description

Merge boundary-sharing (in "Rook" contiguity) grids with fewer target features than the threshold. This function strongly assumes that the input is returned from the internal function `par_make_grid`, which has `"CGRIDID"` as the unique id field.

Usage

```
par_merge_grid(
  points_in = NULL,
  grid_in = NULL,
  grid_min_features = NULL,
  merge_max = 4L
)
```

Arguments

points_in	sf or SpatVector object. Target points of computation.
grid_in	sf or SpatVector object. The grid generated by the internal function par_make_grid.
grid_min_features	integer(1). Threshold to merge adjacent grids.
merge_max	integer(1). Maximum number of grids to merge per merged set. Default is 4. For example, if the number of grids to merge is 20 and merge_max is 10, the function will split the 20 grids into two sets of 10 grids.

Value

A sf or SpatVector object of computation grids.

Note

This function will not work properly if grid_in has more than one million grids.

Author(s)

Insang Song

References

- Polsby DD, Popper FJ. (1991). The Third Criterion: Compactness as a Procedural Safeguard Against Partisan Gerrymandering. *Yale Law & Policy Review*, 9(2), 301–353.

See Also

Other Parallelization: par_cut_coords(), par_grid(), par_grid_mirai(), par_hierarchy(), par_hierarchy_mirai(), par_make_dggrid(), par_make_grid(), par_make_h3(), par_multirasters(), par_multirasters_mirai(), par_pad_balanced(), par_pad_grid(), par_split_list()

Examples

```
lastpar <- par(mfrow = c(1, 1))
library(sf)
library(igraph)
library(dplyr)
library(spatstat.random)
options(sf_use_s2 = FALSE)
```

```
dg <- sf::st_as_sfc(st_bbox(c(xmin = 0, ymin = 0, xmax = 8e5, ymax = 6e5)))
sf::st_crs(dg) <- 5070
dgs <- sf::st_as_sf(st_make_grid(dg, n = c(20, 15)))
dgs$CGRIDID <- seq(1, nrow(dgs))

dg_sample <- sf::st_sample(dg,
  kappa = 5e-9, mu = 15,
  scale = 15000, type = "Thomas"
)
sf::st_crs(dg_sample) <- sf::st_crs(dg)
dg_merged <- par_merge_grid(sf::st_as_sf(dg_sample), dgs, 100)

plot(sf::st_geometry(dg_merged))
par(lastpar)
```

`par_multirasters` *Parallelize spatial computation over multiple raster files*

Description

Large raster files usually exceed the memory capacity in size. This function can be helpful to process heterogenous raster files with homogeneous summary functions. Heterogenous raster files refer to rasters with different spatial extents and resolutions. Cropping a large raster into a small subset even consumes a lot of memory and adds processing time. This function leverages `terra` `SpatRaster` to distribute computation jobs over multiple threads. It is assumed that users have multiple large raster files in their disk, then each file path is assigned to a thread. Each thread will directly read raster values from the disk using C++ pointers that operate in terra functions. For use, it is strongly recommended to use vector data with small and confined spatial extent for computation to avoid out-of-memory error. `y` argument in `fun_dist` will be used as-is. That means no preprocessing or subsetting will be applied. Please be aware of the spatial extent and size of the inputs.

Usage

```
par_multirasters(filenames, fun_dist, ..., .debug = FALSE)
```

Arguments

`filenames`	character. A vector or list of full file paths of raster files. n is the total number of raster files.
`fun_dist`	terra or chopin functions that accept `SpatRaster` object in an argument. In particular, `x` and `y` arguments should be present and `x` should be a `SpatRaster`.
`...`	Arguments passed to the argument `fun_dist`.
`.debug`	logical(1). Default is `FALSE`. If `TRUE` and a unit computation fails, the error message and the file path where the error occurred will be included in the output.

Value

a data.frame object with computation results. For entries of the results, consult the function used in `fun_dist` argument.

Author(s)

Insang Song <geoissong@gmail.com>

See Also

`future::multisession`, `future::multicore`, `future::cluster`, `future.mirai::mirai_multisession`, `future::plan`, `par_convert_f`

Other Parallelization: `par_cut_coords()`, `par_grid()`, `par_grid_mirai()`, `par_hierarchy()`, `par_hierarchy_mirai()`, `par_make_dggrid()`, `par_make_grid()`, `par_make_h3()`, `par_merge_grid()`, `par_multirasters_mirai()`, `par_pad_balanced()`, `par_pad_grid()`, `par_split_list()`

Examples

```
lastpar <- par(mfrow = c(1, 1))
library(terra)
library(sf)
library(future)
library(future.mirai)
options(sf_use_s2 = FALSE)
future::plan(future.mirai::mirai_multisession, workers = 2)

nccnty <- sf::st_read(
  system.file("shape/nc.shp", package = "sf")
)
nccnty <- sf::st_transform(nccnty, "EPSG:5070")

nccntygrid <- sf::st_make_grid(nccnty, n = c(200, 100))
nccntygrid <- sf::st_as_sf(nccntygrid)
nccntygrid$GEOID <- sprintf("%05d", seq_len(nrow(nccntygrid)))
nccntygrid <- sf::st_intersection(nccntygrid, nccnty)

rrast <- terra::rast(nccnty, nrow = 600, ncol = 1320)
terra::values(rrast) <- rgamma(7.92e5, 4, 2)

tdir <- tempdir(check = TRUE)
testpath1 <- file.path(tdir, "test1.tif")
testpath2 <- file.path(tdir, "test2.tif")
terra::writeRaster(rrast, testpath1, overwrite = TRUE)
terra::writeRaster(rrast, testpath2, overwrite = TRUE)
testfiles <- list.files(tdir, pattern = "tif$", full.names = TRUE)

res <- par_multirasters(
  filenames = testfiles,
  fun_dist = extract_at,
  x = testpath1,
  y = nccnty,
```

```
  id = "GEOID",
  func = "mean"
)

future::plan(future::sequential)
mirai::daemons(0)
par(lastpar)
```

`par_multirasters_mirai` *Parallelize spatial computation over multiple raster files*

Description

Large raster files usually exceed the memory capacity in size. This function can be helpful to process heterogenous raster files with homogeneous summary functions. Heterogenous raster files refer to rasters with different spatial extents and resolutions. Cropping a large raster into a small subset even consumes a lot of memory and adds processing time. This function leverages `terra` `SpatRaster` to distribute computation jobs over multiple threads. It is assumed that users have multiple large raster files in their disk, then each file path is assigned to a thread. Each thread will directly read raster values from the disk using C++ pointers that operate in terra functions. For use, it is strongly recommended to use vector data with small and confined spatial extent for computation to avoid out-of-memory error. `y` argument in `fun_dist` will be used as-is. That means no preprocessing or subsetting will be applied. Please be aware of the spatial extent and size of the inputs.

Usage

```
par_multirasters_mirai(filenames, fun_dist, ..., .debug = TRUE)
```

Arguments

`filenames`	character. A vector or list of full file paths of raster files. n is the total number of raster files.
`fun_dist`	terra or chopin functions that accept `SpatRaster` object in an argument. In particular, `x` and `y` arguments should be present and `x` should be a `SpatRaster`.
`...`	Arguments passed to the argument `fun_dist`.
`.debug`	logical(1). Default is `FALSE`. If `TRUE` and a unit computation fails, the error message and the file path where the error occurred will be included in the output.

Value

a data.frame object with computation results. For entries of the results, consult the function used in `fun_dist` argument.

Author(s)

Insang Song <geoissong@gmail.com>

See Also

mirai::mirai, mirai::mirai_map, mirai::daemons, par_convert_f

Other Parallelization: par_cut_coords(), par_grid(), par_grid_mirai(), par_hierarchy(), par_hierarchy_mirai(), par_make_dggrid(), par_make_grid(), par_make_h3(), par_merge_grid(), par_multirasters(), par_pad_balanced(), par_pad_grid(), par_split_list()

Examples

```
lastpar <- par(mfrow = c(1, 1))
library(terra)
library(sf)
library(mirai)
options(sf_use_s2 = FALSE)
mirai::daemons(4)

nccnty <- sf::st_read(
  system.file("shape/nc.shp", package = "sf")
)
nccnty <- sf::st_transform(nccnty, "EPSG:5070")
nccnty <- nccnty[seq_len(30L), ]

nccntygrid <- sf::st_make_grid(nccnty, n = c(200, 100))
nccntygrid <- sf::st_as_sf(nccntygrid)
nccntygrid$GEOID <- sprintf("%05d", seq_len(nrow(nccntygrid)))
nccntygrid <- sf::st_intersection(nccntygrid, nccnty)

rrast <- terra::rast(nccnty, nrow = 600, ncol = 1320)
terra::values(rrast) <- rgamma(7.92e5, 4, 2)

tdir <- tempdir(check = TRUE)
terra::writeRaster(rrast, file.path(tdir, "test1.tif"), overwrite = TRUE)
terra::writeRaster(rrast, file.path(tdir, "test2.tif"), overwrite = TRUE)
testfiles <- list.files(tdir, pattern = "tif$", full.names = TRUE)

res <- par_multirasters_mirai(
  filenames = testfiles,
  fun_dist = extract_at,
  x = rrast,
  y = nccnty,
  id = "GEOID",
  func = "mean"
)
mirai::daemons(0L)
par(lastpar)
```

---

par_pad_balanced *Extension of par_make_balanced for padded grids*

Description

This function utilizes `anticlust::balanced_clustering()` to split the input into equal size subgroups then transform the data to be compatible with the output of `par_pad_grid`, for which a set of padded grids of the extent of input point subsets (as recorded in the field named `"CGRIDID"`) is generated out of input points.

Usage

```
par_pad_balanced(points_in = NULL, ngroups, padding)
```

Arguments

`points_in`	`sf` or `SpatVector` object. Point geometries. Default is NULL.
`ngroups`	integer(1). The number of groups.
`padding`	numeric(1). A extrusion factor to make buffer to clip actual datasets. Depending on the length unit of the CRS of input.

Value

A list of two,

- `original`: exhaustive and non-overlapping grid polygons in the class of input
- padded: a square buffer of each polygon in `original`. Used for computation.

Author(s)

Insang Song

See Also

Other Parallelization: `par_cut_coords()`, `par_grid()`, `par_grid_mirai()`, `par_hierarchy()`, `par_hierarchy_mirai()`, `par_make_dggrid()`, `par_make_grid()`, `par_make_h3()`, `par_merge_grid()`, `par_multirasters()`, `par_multirasters_mirai()`, `par_pad_grid()`, `par_split_list()`

Examples

```
lastpar <- par(mfrow = c(1, 1))
library(terra)
library(sf)
options(sf_use_s2 = FALSE)

ncpath <- system.file("gpkg/nc.gpkg", package = "sf")
nc <- terra::vect(ncpath)
nc_rp <- terra::spatSample(nc, 1000)

nc_gr <- par_pad_balanced(nc_rp, 10L, 1000)
nc_gr
par(lastpar)
```

par_pad_grid *Get a set of computational grids*

Description

Using input points, the bounding box is split to the predefined numbers of columns and rows. Each grid will be buffered by the radius.

Usage

```
par_pad_grid(
  input,
  mode = c("grid", "grid_advanced", "grid_quantile", "h3", "dggrid"),
  nx = 10L,
  ny = 10L,
  grid_min_features = 30L,
  padding = NULL,
  unit = NULL,
  quantiles = NULL,
  merge_max = NULL,
  return_wkt = FALSE,
  res = 8L,
  ...
)
```

Arguments

input	sf or Spat* object.
mode	character(1). Mode of region construction. One of • "grid" (simple grid regardless of the number of features in each grid) • "grid_advanced" (merging adjacent grids with smaller number of features than grid_min_features). The argument grid_min_features should be specified. • "grid_quantile" (x and y quantiles): an argument quantiles should be specified. • "h3" (H3 hexagons): an argument res should be specified. • "dggrid" (DGG grids): an argument res should be specified.
nx	integer(1). The number of grids along x-axis.
ny	integer(1). The number of grids along y-axis.
grid_min_features	integer(1). A threshold to merging adjacent grids
padding	numeric(1). A extrusion factor to make buffer to clip actual datasets. Depending on the length unit of the CRS of input.
unit	character(1). The length unit for padding (optional). units::set_units is used for padding when sf object is used. See link for the list of acceptable unit forms.

quantiles	numeric. Quantiles for grid_quantile mode.
merge_max	integer(1). Maximum number of grids to merge per merged set.
return_wkt	logical(1). Return WKT format. When TRUE, the return value will be a list of two WKT strings.
res	integer(1). Resolution for h3 and dggrid modes.
...	arguments passed to the internal function

Value

A list of two,

- original: exhaustive (filling completely) and non-overlapping grid polygons in the class of input
- padded: a square buffer of each polygon in original. Used for computation.

Author(s)

Insang Song

See Also

Other Parallelization: par_cut_coords(), par_grid(), par_grid_mirai(), par_hierarchy(), par_hierarchy_mirai(), par_make_dggrid(), par_make_grid(), par_make_h3(), par_merge_grid(), par_multirasters(), par_multirasters_mirai(), par_pad_balanced(), par_split_list()

Examples

```
lastpar <- par(mfrow = c(1, 1))
# data
library(sf)
options(sf_use_s2 = FALSE)
ncpath <- system.file("shape/nc.shp", package = "sf")
nc <- read_sf(ncpath)
nc <- st_transform(nc, "EPSG:5070")

# run: nx and ny should strictly be integers
nc_comp_region <-
  par_pad_grid(
    nc,
    mode = "grid",
    nx = 4L, ny = 2L,
    padding = 10000
  )
par(mfcol = c(2, 3))
plot(nc_comp_region$original$geometry, main = "Original grid")
plot(nc_comp_region$padded$geometry, main = "Padded grid")

nc_comp_region_wkt <-
  par_pad_grid(
    nc,
```

```
    mode = "grid",
    nx = 4L, ny = 2L,
    padding = 10000,
    return_wkt = TRUE
  )
nc_comp_region_wkt$original
nc_comp_region_wkt$padded

if (rlang::is_installed("h3r")) {
  suppressWarnings(
    nc_comp_region_h3 <-
      par_pad_grid(
        nc,
        mode = "h3",
        res = 4L,
        padding = 10000
      )
  )
  plot(nc_comp_region_h3$original$geometry, main = "H3 grid (lv.4)")
  plot(nc_comp_region_h3$padded$geometry, main = "H3 padded grid (lv.4)")
}
if (rlang::is_installed("dggridR")) {
  nc_comp_region_dggrid <-
  par_pad_grid(
    nc,
    mode = "dggrid",
    res = 7L,
    padding = 10000
  )
  plot(nc_comp_region_dggrid$original$geometry, main = "DGGRID (lv.7)")
  plot(nc_comp_region_dggrid$padded$geometry, main = "Padded DGGRID (lv.7)")
}
par(lastpar)
```

`par_split_list` *Split grid list to a nested list of row-wise data frames*

Description

Split grid list to a nested list of row-wise data frames

Usage

```
par_split_list(gridlist)
```

Arguments

`gridlist`	list. Output of `par_pad_grid` or `par_pad_balanced`

Details

If the input is a data frame, the function will return a list of two data frames: `original` and `padded`. If the input is a WKT vector, the function will return a list of two WKT strings: `original` and `padded`.

Value

A nested list of data frames or WKT strings.

See Also

Other Parallelization: `par_cut_coords()`, `par_grid()`, `par_grid_mirai()`, `par_hierarchy()`, `par_hierarchy_mirai()`, `par_make_dggrid()`, `par_make_grid()`, `par_make_h3()`, `par_merge_grid()`, `par_multirasters()`, `par_multirasters_mirai()`, `par_pad_balanced()`, `par_pad_grid()`

Examples

```
lastpar <- par(mfrow = c(1, 1))

library(sf)
library(terra)
options(sf_use_s2 = FALSE)

ncpath <- system.file("shape/nc.shp", package = "sf")
nc <- read_sf(ncpath)
nc <- st_transform(nc, "EPSG:5070")
nc_comp_region <-
  par_pad_grid(
    nc,
    mode = "grid",
    nx = 4L, ny = 2L,
    padding = 10000
  )
par_split_list(nc_comp_region)

par(lastpar)
```

---

prediction_grid *Regular grid points in the mainland United States at 1km spatial resolution*

Description

Regular grid points in the mainland United States at 1km spatial resolution

Usage

```
prediction_grid
```

Format

A data frame with 8,092,995 rows and three variables:

site_id Unique point identifier. Arbitrarily generated.

lon Longitude

lat Latitude

Note

Coordinates are in EPSG:5070 (Conus Albers Equal Area)

Source

Mainland United States polygon was obtained from the US Census Bureau.

See Also

Other Dataset: ncpoints

Examples

```
data("prediction_grid", package = "chopin")
```

---

summarize_aw *Area weighted summary using two polygon objects*

Description

When x and y are different classes, poly_weight will be converted to the class of x.

Usage

```
summarize_aw(x, y, ...)

## S4 method for signature 'SpatVector,SpatVector'
summarize_aw(
  x,
  y,
  target_fields = NULL,
  id_x = "ID",
  fun = stats::weighted.mean,
  extent = NULL,
  ...
)

## S4 method for signature 'character,character'
summarize_aw(
```

```
  x,
  y,
  target_fields = NULL,
  id_x = "ID",
  fun = stats::weighted.mean,
  out_class = "terra",
  extent = NULL,
  ...
)

## S4 method for signature 'sf,sf'
summarize_aw(
  x,
  y,
  target_fields = NULL,
  id_x = "ID",
  fun = NULL,
  extent = NULL,
  ...
)
```

Arguments

x	A sf/SpatVector object or file path of polygons detectable with GDAL driver at weighted means will be calculated.
y	A sf/SpatVector object or file path of polygons from which weighted means will be calculated.
...	Additional arguments depending on class of x and y.
target_fields	character. Field names to calculate area-weighted.
id_x	character(1). The unique identifier of each polygon in x. Default is "ID".
fun	function(1)/character(1). The function to calculate the weighted summary. Default is stats::weighted.mean. The function must have a w argument. If both x and y are sf, it should be one of c("sum", "mean"). It will determine extensive argument in sf::st_interpolate_aw.
extent	numeric(4) or SpatExtent object. Extent of clipping x. It only works with x of character(1) file path. See terra::ext for more details. Coordinate systems should match.
out_class	character(1). "sf" or "terra". Output class.

Value

A data.frame with all numeric fields of area-weighted means.

Note

x and y classes should match. If x and y are characters, they will be read as sf objects.

Author(s)

Insang Song <geoissong@gmail.com>

See Also

Other Macros for calculation: extract_at(), kernelfunction(), summarize_pp(), summarize_sedc(), summarize_st()

Examples

```
lastpar <- par(mfrow = c(1, 1))
# package
library(sf)
options(sf_use_s2 = FALSE)
nc <- sf::st_read(system.file("shape/nc.shp", package="sf"))
nc <- sf::st_transform(nc, "EPSG:5070")
pp <- sf::st_sample(nc, size = 300)
pp <- sf::st_as_sf(pp)
pp[["id"]] <- seq(1, nrow(pp))
sf::st_crs(pp) <- "EPSG:5070"
ppb <- sf::st_buffer(pp, nQuadSegs=180, dist = units::set_units(20, "km"))

suppressWarnings(
  ppb_nc_aw <-
    summarize_aw(
      ppb, nc, c("BIR74", "BIR79"),
      "id", fun = "sum"
    )
)
summary(ppb_nc_aw)

# terra examples
library(terra)
ncpath <- system.file("gpkg/nc.gpkg", package = "sf")
nc <- terra::vect(ncpath)
pp <- terra::spatSample(nc, size = 300)
pp[["id"]] <- seq(1, nrow(pp))
ppb <- terra::buffer(pp, 20000)

suppressWarnings(
  ppb_nc_aw <-
    summarize_aw(
      ppb, nc, c("BIR74", "BIR79"), "id",
      fun = sum
    )
)
summary(ppb_nc_aw)
par(lastpar)
```

summarize_pp *Point to polygon summary using target polygons and source points*

Description

When x and y are different classes, y will be converted to the class of x.

Usage

```
summarize_pp(x, y, ...)

## S4 method for signature 'SpatVector'
summarize_pp(
  x,
  y,
  target_fields = NULL,
  id_x = "ID",
  fun = mean,
  extent = NULL,
  ...
)

## S4 method for signature 'character'
summarize_pp(
  x,
  y,
  target_fields = NULL,
  id_x = "ID",
  fun = mean,
  out_class = "terra",
  extent = NULL,
  ...
)

## S4 method for signature 'sf'
summarize_pp(
  x,
  y,
  target_fields = NULL,
  id_x = "ID",
  fun = mean,
  extent = NULL,
  ...
)
```

Arguments

x	A sf/SpatVector object or file path of polygons to which point attributes will be summarized.
y	A sf/SpatVector object or file path of points from which summaries will be calculated.
...	Additional arguments passed to fun.
target_fields	character. Field names in y to summarize.
id_x	character(1). The unique identifier of each polygon in x. Default is "ID".
fun	function(1)/character(1). The function to calculate the point summary. Default is mean. Character input should be a summary function accepted by match.fun().
extent	numeric(4) or SpatExtent object. Extent of clipping x. It only works with x of character(1) file path. See terra::ext for more details. Coordinate systems should match.
out_class	character(1). "sf" or "terra". Output class.

Value

If x is sf, returns an sf object with summarized fields joined to polygons. Otherwise returns a data.frame.

Note

x should contain polygon geometries and y should contain point geometries.

Author(s)

Insang Song <geoissong@gmail.com>

See Also

Other Macros for calculation: extract_at(), kernelfunction(), summarize_aw(), summarize_sedc(), summarize_st()

---

summarize_sedc *Calculate Sum of Exponentially Decaying Contributions (SEDC) covariates*

Description

Calculate Sum of Exponentially Decaying Contributions (SEDC) covariates

Usage

```
summarize_sedc(
  point_from = NULL,
  point_to = NULL,
  id = NULL,
  sedc_bandwidth = NULL,
  threshold = NULL,
  target_fields = NULL,
  extent_from = NULL,
  extent_to = NULL,
  ...
)
```

Arguments

point_from	SpatVector object. Locations where the sum of SEDCs are calculated.
point_to	SpatVector object. Locations where each SEDC is calculated.
id	character(1). Name of the unique id field in point_to.
sedc_bandwidth	numeric(1). Distance at which the source concentration is reduced to exp(-3) (approximately -95 %)
threshold	numeric(1). For computational efficiency, the nearest points in threshold will be selected. 2 * sedc_bandwidth is applied if this value remains NULL.
target_fields	character. Field names to calculate SEDC.
extent_from	numeric(4) or SpatExtent. Extent of clipping point_from. It only works with point_from of character(1) file path. See terra::ext for more details. Coordinate systems should match.
extent_to	numeric(4) or SpatExtent. Extent of clipping point_to.
...	Placeholder.

Details

The SEDC is specialized in vector to vector summary of covariates with exponential decay. Decaying slope will be defined by sedc_bandwidth, where the concentration of the source is reduced to $\exp(-3)$ (approximately 5 \ of the attenuating concentration with the distance from the sources. It can be thought of as a fixed bandwidth kernel weighted sum of covariates, which encapsulates three steps:

- Calculate the distance between each source and target points.
- Calculate the weight of each source point with the exponential decay.
- Summarize the weighted covariates.

Value

data.frame object with input field names with a suffix "_sedc" where the sums of EDC are stored. Additional attributes are attached for the EDC information.

- `attr(result, "sedc_bandwidth")`: the bandwidth where concentration reduces to approximately five percent
- `attr(result, "sedc_threshold")`: the threshold distance at which emission source points are excluded beyond that

Note

Distance calculation is done with `terra` functions internally. Thus, the function internally converts `sf` objects in `point_*` arguments to `terra`. Please note that any `NA` values in the input will be ignored in SEDC calculation.

Author(s)

Insang Song

References

- Messier KP, Akita Y, Serre ML. (2012). Integrating Address Geocoding, Land Use Regression, and Spatiotemporal Geostatistical Estimation for Groundwater Tetrachloroethylene. *Environmental Science & Technology* 46(5), 2772-2780.(doi:10.1021/es203152a)
- Wiesner C. (n.d.). Euclidean Sum of Exponentially Decaying Contributions Tutorial.

See Also

Other Macros for calculation: `extract_at()`, `kernelfunction()`, `summarize_aw()`, `summarize_pp()`, `summarize_st()`

Examples

```
library(terra)
library(sf)
set.seed(101)
ncpath <- system.file("gpkg/nc.gpkg", package = "sf")
nc <- terra::vect(ncpath)
nc <- terra::project(nc, "EPSG:5070")
pnt_from <- terra::centroids(nc, inside = TRUE)
pnt_from <- pnt_from[, "NAME"]
pnt_to <- terra::spatSample(nc, 100L)
pnt_to$pid <- seq(1, 100)
pnt_to <- pnt_to[, "pid"]
pnt_to$val1 <- rgamma(100L, 1, 0.05)
pnt_to$val2 <- rgamma(100L, 2, 1)

vals <- c("val1", "val2")
suppressWarnings(
  summarize_sedc(pnt_from, pnt_to, "NAME", 1e5, 2e5, vals)
)
```

summarize_st *Summarize Data by Time or Space*

Description

Summarize numeric columns of a data frame by either time intervals or spatial features. This function supports two modes of operation: time-based grouping and spatial grouping.

Usage

```
summarize_st(df, f, id_col, .by, ...)
```

Arguments

df	A data frame, sf object, or SpatVector containing the data to summarize.
f	A function to apply for summarization (e.g., "mean", "sum").
id_col	A column name or expression specifying the ID column for grouping.
.by	Either a character string specifying a time unit ("minutes", "hours", "days", "weeks", "months", "quarters", "years") for temporal grouping, or an sf/SpatVector object for spatial grouping.
...	Additional arguments passed to summarize_aw() when doing spatial grouping.

Details

When .by is a character string, the function performs time-based summarization: it groups data by ID and time intervals, then applies the summarization function to all numeric columns.

When .by is an sf or SpatVector object, the function performs spatial summarization using summarize_aw() to aggregate data across spatial features.

Value

A data frame with summarized values. For time-based grouping, includes the original time column and grouped summaries. For spatial grouping, returns the result from summarize_aw().

Note

For time-based grouping, the function expects exactly one time column in the data frame. The time values will be "floored" to the nearest time unit defined by .by for grouping.

See Also

Other Macros for calculation: extract_at(), kernelfunction(), summarize_aw(), summarize_pp(), summarize_sedc()

Index