Package: chopin (via r-universe)

November 20, 2024

Title Computation of Spatial Data by Hierarchical and Objective Partitioning of Inputs for Parallel Processing

Version 0.9.0

Description Geospatial data computation is parallelized by grid, hierarchy, or raster files. Based on future and mirai parallel backends, terra and 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, 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

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

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

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

VignetteBuilder knitr

Config/testthat/edition 3

Config/pak/sysreqs cmake libgdal-dev gdal-bin libgeos-dev libglpk-dev libxml2-dev libnetcdf-dev libssl-dev libproj-dev libsqlite3-dev libudunits2-dev xz-utils

Repository https://ropensci.r-universe.dev

RemoteUrl https://github.com/ropensci/chopin

RemoteRef main

RemoteSha 42799ee62c2acf8a4e45b7e0c63af8dc6e1fb894

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_sedc()

Examples

```
ncpath <- system.file("gpkg/nc.gpkg", package = "sf")
rastpath <- system.file("extdata/nc_srtm15_otm.tif", package = "chopin")

nc <- terra::vect(ncpath)
nc <- terra::project(nc, "EPSG:5070")
rrast <- terra::rast(nc, nrow = 100, ncol = 220)
ncr <- terra::rasterize(nc, rrast)
terra::values(rrast) <- rgamma(2.2e4, 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")
extract_at(
  rrast, ncpath, "NAME", "mean",
  kernel = "epanechnikov",
  bandwidth = 1e5
)
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 arguments to the desired names*

Description

This function maps the arguments of a target function to the desired names. Users will use a named list name_match to standardize the argument names, at least 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, arg_map)
```

Arguments

fun	A function to map arguments.
arg_map	named character vector. c(x = "a", y = "i") will map a and i in fun to x and y, respectively.

Value

Function with arguments mapped.

Note

arg_map should be defined carefully according to the characteristics of fun. After mapping x and y, the resultant function will fail if there remain arguments without default. It is recommended to map all the arguments in fun to avoid any side effects.

Examples

```
cov_map <- arg_mapping <- c(x = "a", y = "b", z = "c", w = "d")
# Example original function
f1 <- function(a, b, c, d) {
  return(a + b + c + d)
}
# Mapping of new argument names to original argument names
arg_mapping <- c(x = "a", y = "b", z = "c", w = "d")
f2 <- par_convert_f(f1, arg_mapping)

# demonstrate f2 with the mapped arguments
f2(x = 1, y = 2, z = -1, w = 10)
```

---

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, but please be advised that some spatial operations do not necessarily give the exact result from what would have been done 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_mirai(), par_hierarchy(), par_hierarchy_mirai(), par_make_grid(), par_merge_grid(), par_multirasters(), par_multirasters_mirai(), par_pad_balanced(), par_pad_grid(), par_split_list()

Examples

```
library(sf)
library(future)
library(future.mirai)
plan(mirai_multisession, workers = 2)
ncpath <- system.file("shape/nc.shp", package = "sf")
ncpoly <- sf::st_read(ncpath)
# sf object
ncpnts <-
  readRDS(
    system.file("extdata/nc_random_point.rds", package = "chopin")
  )
# file path
ncelev <-
    system.file("extdata/nc_srtm15_otm.tif", package = "chopin")

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 = ncelev,
    y = ncpnts,
```

```
  qsegs = 90L,
  radius = 5e3L,
  id = "pid"
)
```

`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`, but please be advised that some spatial operations do not necessarily give the exact result from what would have been done 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

mirai::daemons, mirai::mirai_map, par_convert_f

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

Examples

```
library(sf)
library(mirai)
daemons(4, dispatcher = "process")
ncpath <- system.file("shape/nc.shp", package = "sf")
ncpoly <- sf::st_read(ncpath)
# sf object
ncpnts <-
  readRDS(
    system.file("extdata/nc_random_point.rds", package = "chopin")
  )
# file path
ncelev <-
    system.file("extdata/nc_srtm15_otm.tif", package = "chopin")

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 = ncelev,
    y = ncpnts,
    qsegs = 90L,
    radius = 5e3L,
    id = "pid"
  )
mirai::daemons(0L)
```

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. Please 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, but please 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_grid(), par_merge_grid(), par_multirasters(), par_multirasters_mirai(), par_pad_balanced(), par_pad_grid(), par_split_list()

Examples

```
library(terra)
library(sf)
library(future)
library(future.mirai)
options(sf_use_s2 = FALSE)
future::plan(future.mirai::mirai_multisession, workers = 2)

ncpath <- system.file("extdata/nc_hierarchy.gpkg", package = "chopin")
nccnty <- sf::st_read(ncpath, layer = "county")
nctrct <- sf::st_read(ncpath, layer = "tracts")
ncelev <-
  system.file("extdata/nc_srtm15_otm.tif", package = "chopin")

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 = "GEOID",
    fun_dist = extract_at,
    y = nctrct,
    x = ncelev,
    id = "GEOID",
    func = "mean"
  )
```

`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. Please 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`, but please 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

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

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

Examples

```
library(terra)
library(sf)
library(mirai)
options(sf_use_s2 = FALSE)
mirai::daemons(4, dispatcher = "process")

ncpath <- system.file("extdata/nc_hierarchy.gpkg", package = "chopin")
nccnty <- sf::st_read(ncpath, layer = "county")
nctrct <- sf::st_read(ncpath, layer = "tracts")
ncelev <-
  system.file("extdata/nc_srtm15_otm.tif", package = "chopin")

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 = "GEOID",
    fun_dist = extract_at,
    y = nctrct,
    x = ncelev,
    id = "GEOID",
    func = "mean",
    .debug = TRUE
  )
mirai::daemons(0L)
```

`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 `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_grid()`, `par_multirasters()`, `par_multirasters_mirai()`, `par_pad_balanced()`, `par_pad_grid()`, `par_split_list()`

Examples

```
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_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_grid(), par_merge_grid(), par_multirasters_mirai(), par_pad_balanced(), par_pad_grid(), par_split_list()

Examples

```
library(terra)
library(sf)
library(future)
library(future.mirai)
sf::sf_use_s2(FALSE)
future::plan(future.mirai::mirai_multisession, workers = 2)

ncpath <- system.file("extdata/nc_hierarchy.gpkg", package = "chopin")
nccnty <- sf::st_read(ncpath, layer = "county")
ncelev <-
  system.file("extdata/nc_srtm15_otm.tif", package = "chopin")
ncelevras <- terra::rast(ncelev)

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

```
res <- par_multirasters(
  filenames = testfiles,
  fun_dist = extract_at,
  x = ncelev,
  y = nccnty,
  id = "GEOID",
  func = "mean"
)
```

`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_grid(), par_merge_grid(), par_multirasters(), par_pad_balanced(), par_pad_grid(), par_split_list()

Examples

```
library(terra)
library(sf)
library(mirai)
sf::sf_use_s2(FALSE)
mirai::daemons(4, dispatcher = "process")

ncpath <- system.file("extdata/nc_hierarchy.gpkg", package = "chopin")
nccnty <- sf::st_read(ncpath, layer = "county")
ncelev <-
  system.file("extdata/nc_srtm15_otm.tif", package = "chopin")
ncelevras <- terra::rast(ncelev)

tdir <- tempdir(check = TRUE)
terra::writeRaster(ncelevras, file.path(tdir, "test1.tif"), overwrite = TRUE)
terra::writeRaster(ncelevras, 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 = ncelev,
  y = nccnty,
  id = "GEOID",
  func = "mean"
)
mirai::daemons(0L)
```

---

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_grid(), par_merge_grid(), par_multirasters(), par_multirasters_mirai(), par_pad_grid(), par_split_list()

Examples

```
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_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"),
  nx = 10L,
  ny = 10L,
  grid_min_features = 30L,
  padding = NULL,
  unit = NULL,
  quantiles = NULL,
  merge_max = NULL,
  return_wkt = FALSE,
  ...
)
```

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.
`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.
...	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_grid(), par_merge_grid(), par_multirasters(), par_multirasters_mirai(), par_pad_balanced(), par_split_list()

Examples

```
# 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(1, 2))
plot(nc_comp_region$original$geometry)
plot(nc_comp_region$padded$geometry)

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
```

`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_grid()`, `par_merge_grid()`, `par_multirasters()`, `par_multirasters_mirai()`, `par_pad_balanced()`, `par_pad_grid()`

Examples

```
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)
```

`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_sedc()

Examples

```
# package
library(sf)
options(sf_use_s2 = FALSE)
nc <- sf::st_read(system.file("shape/nc.shp", package="sf"))
nc <- sf::st_transform(nc, 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"))

system.time(
  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")
elev <- system.file("ex/elev.tif", package = "terra")
nc <- terra::vect(ncpath)
elev <- terra::rast(elev)
pp <- terra::spatSample(nc, size = 300)
pp <- terra::project(pp, crs(elev))
pp <- terra::as.points(pp)
pp[["id"]] <- seq(1, nrow(pp))
ppb <- terra::buffer(pp, 20000)

system.time(
  ppb_nc_aw <-
    summarize_aw(
      ppb, nc, c("BIR74", "BIR79"), "id",
      fun = sum
    )
)
summary(ppb_nc_aw)
```

`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.
- Wiesner C. (n.d.). Euclidean Sum of Exponentially Decaying Contributions Tutorial.

See Also

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

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")
summarize_sedc(pnt_from, pnt_to, "NAME", 1e5, 2e5, vals)
```

Index