Package 'MultiscaleSCP'

May 7, 2026

Title Multiscale Systematic Conservation Planning Across Nested H3 Grids

Version 0.1.1

Maintainer Pablo Merlo <pablo.merlo@universityofgalway.ie>

Description Provides tools for multiscale systematic conservation planning using the H3 hierarchical hexagonal grid system (Uber Technologies (2024) <https://h3geo.org>) and the 'prioritizr' package (Hanson et al. (2025) <doi:10.1111/cobi.14376>). Supports the definition and solution of conservation problems across nested H3 resolutions with resolution-specific features, costs, and management attributes, including cross-scale connectivity penalties derived from parent-child relationships. Also includes utilities to evaluate solutions using multiscale-aware diagnostics and to post-process optimization outputs into alternative area-targeted conservation scenarios.

Depends R (>= 4.2)

Imports sf, terra, raster, dplyr, tibble, Matrix, methods, prioritizr, exactextractr, stats, cli, R6, assertthat, h3jsr, rlang

Suggests ggplot2, rnaturalearth, rnaturalearthdata, rmapshaper, knitr, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.3

Config/testthat/edition 3

NeedsCompilation no

Author Pablo Merlo [aut, cre] (ORCID: <https://orcid.org/0000-0002-0626-4993>), Oisín Callery [aut] (ORCID: <https://orcid.org/0000-0002-3388-0951>), Carlos Tighe [ctb], Anthony Grehan [ctb]

Repository CRAN

Date/Publication 2026-03-30 17:00:02 UTC

Contents

`add_multiscale_connectivity_penalties` *Add multiscale connectivity penalties to a prioritizr problem*

Description

This function mirrors `prioritizr::add_connectivity_penalties()` but adds a second, independent symmetric penalty for cross-resolution (vertical) connectivity between H3 planning units. It accepts the same input formats (matrix, `Matrix::dgCMatrix`, data frame, or 4D array) and internally converts them to a sparse connectivity matrix.

Usage

```
add_multiscale_connectivity_penalties(
  x,
  penalty,
  zones = diag(prioritizr::number_of_zones(x)),
  data,
  normalize = c("none", "sym")
)

## S4 method for signature 'ANY,ANY,ANY,matrix,character'
add_multiscale_connectivity_penalties(
  x,
  penalty,
  zones = diag(prioritizr::number_of_zones(x)),
  data,
  normalize = c("none", "sym")
)
```

```
## S4 method for signature 'ANY,ANY,ANY,Matrix,character'
add_multiscale_connectivity_penalties(
  x,
  penalty,
  zones = diag(prioritizr::number_of_zones(x)),
  data,
  normalize = c("none", "sym")
)

## S4 method for signature 'ANY,ANY,ANY,data.frame,character'
add_multiscale_connectivity_penalties(
  x,
  penalty,
  zones = diag(prioritizr::number_of_zones(x)),
  data,
  normalize = c("none", "sym")
)

## S4 method for signature 'ANY,ANY,ANY,dgCMatrix,character'
add_multiscale_connectivity_penalties(
  x,
  penalty,
  zones = diag(prioritizr::number_of_zones(x)),
  data,
  normalize = c("none", "sym")
)

## S4 method for signature 'ANY,ANY,ANY,array,character'
add_multiscale_connectivity_penalties(
  x,
  penalty,
  zones = diag(prioritizr::number_of_zones(x)),
  data,
  normalize = c("none", "sym")
)
```

Arguments

x	problem() object.
penalty	numeric penalty that is used to scale the importance of selecting planning units with strong connectivity between them compared to the main problem objective (e.g., solution cost when the argument to x has a minimum set objective set using add_min_set_objective()). Higher penalty values can be used to obtain solutions with a high degree of connectivity, and smaller penalty values can be used to obtain solutions with a small degree of connectivity. Note that negative penalty values can be used to obtain solutions that have very little connectivity.
zones	matrix or Matrix object describing the level of connectivity between different

zones. Each row and column corresponds to a different zone in the argument to x, and cell values indicate the level of connectivity between each combination of zones. Cell values along the diagonal of the matrix represent the level of connectivity between planning units allocated to the same zone. Cell values must lay between 1 and -1, where negative values favor solutions with weak connectivity. The default argument to zones is an identity matrix (i.e., a matrix with ones along the matrix diagonal and zeros elsewhere), so that planning units are only considered to be connected when they are allocated to the same zone. This argument is required when working with multiple zones and the argument to data is a matrix or Matrix object. If the argument to data is an array or data.frame with data for multiple zones (e.g., using the "zone1" and "zone2" column names), this argument must explicitly be set to NULL otherwise an error will be thrown.

data — A symmetric connectivity object (matrix, Matrix::dgCMatrix, data frame in Marxan format, or 4D array) describing cross-scale links.

normalize — Either "none" (use data as provided) or "sym" to apply symmetric degree normalization before penalization.

Value

The modified ConservationProblem object.

Functions

- add_multiscale_connectivity_penalties(x = ANY, penalty = ANY, zones = ANY, data = matrix, normalize = character): matrix method
- add_multiscale_connectivity_penalties(x = ANY, penalty = ANY, zones = ANY, data = Matrix, normalize = character): Matrix method
- add_multiscale_connectivity_penalties(x = ANY, penalty = ANY, zones = ANY, data = data.frame, normalize = character): data.frame (Marxan) method
- add_multiscale_connectivity_penalties(x = ANY, penalty = ANY, zones = ANY, data = dgCMatrix, normalize = character): dgCMatrix method
- add_multiscale_connectivity_penalties(x = ANY, penalty = ANY, zones = ANY, data = array, normalize = character): array (4D) method

---

build_crossscale_index

Build cross-scale index structures for H3-based SCP workflows

---

Description

Extends the basic hierarchy from build_h3_maps() into full ancestor, descendant and resolution-index mappings used by all multiscale selection and evaluation functions.

Usage

```
build_crossscale_index(maps)
```

Arguments

maps	The list returned by build_h3_maps().

Value

A list with elements:

- res_levels
- rows_by_res
- pos_in_res
- anc_at_res
- desc_at_res
- finer_rows_by_r0cell

Examples

```
h3_child  <- "8a2a1072b59ffff"
h3_parent <- "872a1072bffffff"

maps <- build_h3_maps(
  s_or_h3 = c(h3_parent, h3_child),
  res_vec = c(7L, 8L)
)
cs_idx <- build_crossscale_index(maps)

names(cs_idx)
cs_idx$res_levels
```

---

build_h3_maps *Build H3 hierarchy maps for multiscale planning units*

Description

Creates basic parent–child relationships for a multiresolution H3 planning-unit dataset. This is the **core structure** used by all cross-scale operations.

Usage

```
build_h3_maps(s_or_h3, res_vec = NULL, res_levels = NULL)
```

Arguments

s_or_h3	Either an sf object with h3_address and res columns, or a character vector of H3 indexes.
res_vec	Optional integer vector of resolutions if s_or_h3 is not an sf.
res_levels	Optional integer vector of reporting resolutions.

Value

A list with elements h3_vec, res_vec, res_levels, row_idx_by_h3, nearest_parent_row_of, and children_by_row.

Examples

```
# Minimal example: two resolutions, parent-child relationship
h3_child  <- "8a2a1072b59ffff" # example H3 index
h3_parent <- "872a1072bffffff"

maps <- build_h3_maps(
  s_or_h3  = c(h3_parent, h3_child),
  res_vec  = c(7L, 8L),
  res_levels = c(7L, 8L)
)

str(maps, max.level = 1)
maps$nearest_parent_row_of
```

---

build_multiscale_connectivity_matrix

Build a multiscale H3 connectivity matrix

Description

Construct a symmetric sparse connectivity matrix linking each finer H3 planning unit to its parent at the next coarser resolution. This is typically used as the data input to add_multiscale_connectivity_penalties().

Usage

```
build_multiscale_connectivity_matrix(maps, symmetric = TRUE)
```

Arguments

maps	A list as returned by build_h3_maps(), containing at least h3_vec, res_vec, res_levels, and row_idx_by_h3.
symmetric	Logical; if TRUE (default), the matrix is symmetrised so that each parent–child link appears in both directions.

Value

A `Matrix::dgCMatrix` connectivity matrix of size `n_pu × n_pu`.

Examples

```
# Minimal 2-resolution parent-child example
h3_child  <- "8a2a1072b59ffff"
h3_parent <- "872a1072bffffff"

maps <- build_h3_maps(
  s_or_h3 = c(h3_parent, h3_child),
  res_vec = c(7L, 8L)
)

conn <- build_multiscale_connectivity_matrix(maps)
conn
```

`compute_overlaps_by_resolution` *Count cross-resolution overlaps in a selection*

Description

For a given 0/1 selection column, counts how many planning units are simultaneously selected at a finer resolution and at their ancestor cell at a coarser resolution (using the hierarchy in `maps`).

For each pair of resolutions `r_low` < `r_high`, it reports the number of co-selected ancestor–descendant pairs, plus a total across all pairs.

Usage

```
compute_overlaps_by_resolution(
  s,
  flag_col = "selected_by_resolution_final",
  maps
)
```

Arguments

`s`	An `sf` or data frame of planning units. Must contain a resolution column `res` and a 0/1 selection column referenced by `flag_col`.
`flag_col`	Name of the 0/1 selection column to analyse (e.g. `"selected_by_resolution"`).
`maps`	A hierarchy list as returned by `build_h3_maps()`, containing at least `res_levels` and `nearest_parent_row_of`.

Value

A named integer vector with one entry per resolution pair (e.g. `"5-6"`, `"6-7"`) and a `"total"` element.

Examples

```
parent <- "872a1072bffffff"
kids   <- c("882a1072b1fffff", "882a1072b3fffff")

s <- data.frame(
  h3_address = c(parent, kids),
  res = c(7L, 8L, 8L),
  selected_by_resolution_final = c(1L, 1L, 0L)
)
maps <- build_h3_maps(s, res_levels = c(7L, 8L))

compute_overlaps_by_resolution(s, "selected_by_resolution_final", maps)
```

---

compute_pu_scores *Compute importance scores for planning units*

Description

This function calculates a continuous importance score (`ensemble_score`) for each planning unit by combining two sources of information: (1) a rarity-weighted aggregation of feature values, and (2) how consistently each planning unit is selected across one or more solutions.

The relative influence of these components is controlled by `alpha_freq`. When multiple solutions are provided, selection consistency reflects how often each planning unit is selected across the portfolio. When a single solution is provided, it reflects whether the planning unit is selected or not. If no solution columns are present, the score is based entirely on the rarity-weighted feature component.

Optionally, `locked_in` planning units can be forced to rank above all others.

Usage

```
compute_pu_scores(
  s,
  feature_mat,
  feature_weights = NULL,
  alpha_freq = 0.25,
  freq_gamma = 1.5,
  lock_mode = c("top", "none"),
  winsor_p = NULL,
  cost_col = NULL,
  cost_beta = 1
)
```

Arguments

s	An sf object with one row per planning unit. Must contain any solution_* columns used to compute selection consistency, and optionally a logical locked_in column.
feature_mat	A numeric matrix or data frame with one row per planning unit and one column per feature (e.g. proportion of each feature in each PU).
feature_weights	Optional named numeric vector of user weights for the features. Names must match colnames(feature_mat). If NULL, all features receive equal weight.
alpha_freq	Numeric in $[0, 1]$ controlling the trade-off between feature score and selection consistency. Values near 0 emphasize features, values near 1 emphasize consistency across solutions.
freq_gamma	Numeric exponent applied to selection consistency (after percentile ranking) to emphasize planning units that are selected in many solutions (freq_gamma > 1 increases contrast).
lock_mode	Character, either "none" (locked-in PUs are treated like any others, aside from their higher selection consistency) or "top" (locked-in PUs are always ranked above non-locked ones).
winsor_p	Optional numeric in $[0, 0.5)$. If provided, feature values are winsorized at the winsor_p and 1 - winsor_p quantiles prior to rank/ECDF scaling to $[0, 1]$. If NULL (default), no winsorization is performed.
cost_col	Optional name of a numeric column in s containing planning-unit costs. If provided, the feature-based score is divided by cost^cost_beta prior to final rescaling.
cost_beta	Numeric exponent applied to costs when cost_col is provided. Defaults to 1 (benefit per cost).

Value

The same sf object s with two new numeric columns: selection_consistency and ensemble_score.

Examples

```
# Minimal sf with 3 planning units + two solution columns
s <- sf::st_as_sf(
  data.frame(
    x = c(0, 1, 2),
    y = c(0, 0, 0),
    solution_1 = c(1, 0, 1),
    solution_2 = c(1, 1, 0),
    locked_in  = c(FALSE, TRUE, FALSE)
  ),
  coords = c("x", "y"), crs = 4326
)

# Feature matrix must match nrow(s)
feature_mat <- data.frame(
```

```
  f1 = c(0.2, 0.0, 0.8),
  f2 = c(0.0, 0.5, 0.1)
)

s2 <- compute_pu_scores(
  s, feature_mat,
  alpha_freq = 0.25,
  lock_mode  = "top"
)

s2[, c("selection_consistency", "ensemble_score", "locked_in")]
```

`compute_pu_scores_crossscale` *Compute cross-scale planning-unit scores by resolution*

Description

This function computes a feature-based importance score at each resolution present in a multiscale planning-unit set, and then propagates those resolution-specific scores through the H3 hierarchy so every planning unit receives a score "as seen from" each resolution.

Propagation rules:

- **Downward (parent → children):** undamped broadcast. All descendants of a given parent at resolution `r` inherit the same `score_from_r` value.
- **Upward (children → parent):** aggregation of descendant scores at resolution `r` using `mean` (default) or `max`.

Usage

```
compute_pu_scores_crossscale(
  s,
  feature_mat,
  maps,
  cs_idx,
  feature_weights = NULL,
  winsor_p = NULL,
  cost_col = NULL,
  cost_beta = 1,
  agg_mode = c("mean", "max"),
  res_col = "res",
  res_levels = NULL,
  feature_cols_by_res = NULL,
  prefix = "score_from_r"
)
```

Arguments

s	An sf object with one row per planning unit. Must contain a resolution column (default "res").
feature_mat	A numeric matrix/data.frame with one row per planning unit and one column per feature. Feature columns are expected to be prefixed by resolution (e.g., r5_, r6_, ...), unless feature_cols_by_res is supplied.
maps	Output of build_h3_maps().
cs_idx	Output of build_crossscale_index().
feature_weights	Optional named numeric vector of user weights for features. Names must match colnames(feature_mat).
winsor_p	Optional winsorization level passed to the feature-only score computation. See compute_pu_scores().
cost_col	Optional name of a numeric cost column in s. If provided, feature-based scores are divided by cost^cost_beta within each resolution before propagation.
cost_beta	Numeric exponent applied to costs when cost_col is provided.
agg_mode	Character, aggregation used for upward propagation (children → parent). One of "mean" or "max".
res_col	Name of the resolution column in s. Defaults to "res".
res_levels	Optional integer vector of resolutions to compute. Defaults to maps$res_levels.
feature_cols_by_res	Optional named list mapping each resolution (as a character) to a character vector of feature column names. If NULL, columns are inferred using the prefix pattern ^r{res}_.
prefix	Column name prefix for outputs. Defaults to "score_from_r".

Value

The same sf object s with one new numeric column per resolution, named paste0(prefix, res) (e.g., score_from_r5).

Examples

```
# Tiny 2-level setup: 2 parents (r7), and 2 children (r8) under the first parent
parent1 <- "872a1072bffffff"
parent2 <- "872a10729ffffff"
kids    <- c("882a1072b1fffff", "882a1072b3fffff")

h3_vec  <- c(parent1, parent2, kids)
res_vec <- c(7L, 7L, 8L, 8L)

s <- sf::st_as_sf(
  data.frame(
    h3_address = h3_vec,
    res        = res_vec,
    cost       = c(1, 1, 1, 1),
```

```
    x          = c(0, 1, 2, 3),
    y          = c(0, 0, 0, 0)
  ),
  coords = c("x", "y"), crs = 4326
)

# Feature columns prefixed by resolution (r7_, r8_)
feature_mat <- data.frame(
  r7_f1 = c(1.0, 0.2, 0.0, 0.0),
  r8_f1 = c(0.0, 0.0, 0.6, 0.2)
)

maps   <- build_h3_maps(s)
cs_idx <- build_crossscale_index(maps)

s2 <- compute_pu_scores_crossscale(
  s, feature_mat,
  maps = maps, cs_idx = cs_idx,
  agg_mode = "mean",
  res_col = "res"
)

s2[, c("res", "score_from_r7", "score_from_r8")]
```

`compute_selection_by_resolution` *Stratified multiscale selection by resolution*

Description

Select planning units to meet area-based targets at each H3 resolution. Selection is based on a single score column (e.g., `compute_pu_scores`) and a cross-scale index produced by `build_crossscale_index`.

Resolutions are processed from finest to coarsest. At each resolution, the function accounts for area already covered by selected finer units to avoid double-counting. To keep selections non-overlapping, it prefers units that do not have finer descendants in the input dataset; coarser units with descendants are only used if needed to reach the target.

The result is a resolution-stratified selection that meets per-resolution area targets while minimizing overlap between coarse and fine planning units.

Usage

```
compute_selection_by_resolution(
  s,
  cs_idx,
  target = 0.3,
  score_col = "ensemble_score",
  area_col = "area_km2",
```

```
  res_col = "res",
  out_col = "selected_by_resolution",
  blocked_col = NULL,
  target_by_res = NULL
)
```

Arguments

s	An sf object or data frame of planning units. Must contain at least the columns referenced by score_col, area_col, and res_col.
cs_idx	A cross-scale index list as returned by build_crossscale_index.
target	Default area-based target proportion (e.g. 0.3) applied to all resolutions if target_by_res is NULL.
score_col	Name of the column in s containing the planning-unit score used to rank candidates (e.g. "ensemble_score").
area_col	Name of the column in s containing planning-unit areas (e.g. "area_km2").
res_col	Name of the column in s containing integer resolution codes (e.g. H3 resolution).
out_col	Name of the output column created in s containing the final 0/1 selection flag (default "selected_by_resolution").
blocked_col	Optional name of a column to store a global "has finer descendants" flag as 0/1 (1 = has finer descendants). If NULL (default), this column is not written.
target_by_res	Optional named numeric vector of per-resolution targets, e.g. c("5" = 0.30, "6" = 0.20, "7" = 0.15). Names should match the values in res_col. When supplied, these override target for the corresponding resolutions.

Value

The input s with an additional column named by out_col (0/1), and optionally a column named by blocked_col if requested.

Examples

```
# Build a tiny multiscale set: 2 parents (r7), 2 children (r8) under parent1
parent1 <- "872a1072bffffff"
parent2 <- "872a10729ffffff"
kids1   <- c("882a1072b1fffff", "882a1072b3fffff")

h3_vec  <- c(parent1, parent2, kids1)
res_vec <- c(7L, 7L, 8L, 8L)

s <- data.frame(
  h3_address     = h3_vec,
  res            = res_vec,
  area_km2       = c(14, 14, 2, 2),
  ensemble_score = c(0.2, 0.9, 0.8, 0.1)
)
```

```
maps   <- build_h3_maps(s, res_levels = c(7L, 8L))
cs_idx <- build_crossscale_index(maps)

out <- compute_selection_by_resolution(
  s, cs_idx,
  target    = 0.5,
  score_col = "ensemble_score",
  area_col  = "area_km2",
  res_col   = "res",
  out_col   = "selected_by_resolution"
)

out[, c("res", "area_km2", "ensemble_score", "selected_by_resolution")]
```

`compute_selection_by_strata` *Stratified multiscale selection by strata and resolution*

Description

Select planning units to meet area-based targets for each strata (e.g. country) at each H3 resolution. Selection is based on a single score column (typically `ensemble_score`) and a cross-scale index produced by `build_crossscale_index`.

Resolutions are processed from finest to coarsest. At each strata and resolution, the function accounts for area already covered by selected planning units at other resolutions to avoid double-counting. Within each strata, it prefers native-resolution units that do not have finer descendants in the input dataset, using coarser units with descendants only when needed to reach the target.

The result is a single 0/1 selection that meets per-strata area targets while minimizing cross-scale overlap.

Usage

```
compute_selection_by_strata(
  s,
  cs_idx,
  strata_masks,
  target = 0.3,
  admin_strata = "admin",
  score_col = "ensemble_score",
  area_col = "area_km2",
  res_col = "res",
  out_col = "selected_by_admin",
  blocked_col = NULL,
  target_by_strata = NULL
)
```

Arguments

s	An sf object or data frame of planning units. Must contain at least the columns referenced by admin_strata, score_col, area_col, and res_col.
cs_idx	A cross-scale index list as returned by build_crossscale_index(), containing at least res_levels, rows_by_res, anc_at_res, desc_at_res, and finer_rows_by_r0cell.
strata_masks	Optional named list of logical vectors (length nrow(s)) defining the "in-scope" units for each strata. Names should match the values in admin_strata. If a given strata is missing from strata_masks, a fallback mask admin_strata == A is used.
target	Default area-based target proportion (e.g. 0.3) applied to all strata if target_by_strata is NULL or does not contain an entry for that strata.
admin_strata	Name of the column in s containing strata labels (e.g. "admin").
score_col	Name of the column in s containing the PU score to rank candidates.
area_col	Name of the column in s with PU areas.
res_col	Name of the column in s with integer resolution codes.
out_col	Name of the output column that will be created in s containing the final 0/1 selection flag (default "selected_by_admin").
blocked_col	Optional name of a column to store the global "has finer descendants" flag as 0/1. If NULL (default), this column is not written.
target_by_strata	Optional named numeric vector of per-strata targets, e.g. c("Ireland" = 0.30, "Norway" = 0.25). Names should match the values in admin_strata. When present, these override target for the corresponding strata.

Value

The input s with an additional column out_col (0/1), and optionally blocked_col if requested.

Examples

```
# Tiny multiscale example with two strata (A and B)
parent1 <- "872a1072bffffff"
kids1   <- c("882a1072b1fffff", "882a1072b3fffff")
parent2 <- "872a10729ffffff"
parent3 <- "872a10774ffffff"

h3_vec  <- c(parent1, parent2, parent3, kids1)
res_vec <- c(7L, 7L, 7L, 8L, 8L)

s <- data.frame(
  h3_address     = h3_vec,
  res            = res_vec,
  admin          = c("A", "B", "B", "A", "A"),
  area_km2       = c(14, 14, 14, 2, 2),
  ensemble_score = c(0.2, 0.9, 0.2, 0.8, 0.1)
)
```

```
maps   <- build_h3_maps(s, res_levels = c(7L, 8L))
cs_idx <- build_crossscale_index(maps)

strata_masks <- list(
  A = (s$admin == "A"),
  B = (s$admin == "B")
)

out <- compute_selection_by_strata(
  s, cs_idx,
  strata_masks     = strata_masks,
  target           = 0.3,
  admin_strata     = "admin",
  score_col        = "ensemble_score",
  area_col         = "area_km2",
  res_col          = "res",
  out_col          = "selected_by_admin",
  target_by_strata = c(A = 0.12, B = 0.5)
)

out[, c("admin", "res", "area_km2", "ensemble_score", "selected_by_admin")]
```

`deduplicate_h3_selections` *Deduplicate multiscale selections across H3 resolutions*

Description

Removes redundant planning-unit selections across nested H3 resolutions. Because finer and coarser H3 cells overlap perfectly (parent–child hierarchy), a selection at one resolution makes selections at other resolutions redundant.

This function enforces a consistent hierarchy using one of two strategies:

- `"coarser_first"` – keep coarser selected cells and drop all selected descendants (finer cells). Useful if coarse-scale representation should dominate.
- `"finer_first"` – keep finer selected cells and drop selected ancestors (coarser cells). Useful when fine-scale detail should dominate and coarse cells should not "override" them.

Usage

```
deduplicate_h3_selections(
  s,
  sel_col,
  h3_vec,
  res_vec,
  res_levels,
  nearest_parent_row_of,
```

```
  children_by_row,
  mode = c("coarser_first", "finer_first")
)
```

Arguments

s	An sf or data frame containing at least the selection column (sel_col).
sel_col	Name of the 0/1 column to deduplicate (e.g. "solution_1").
h3_vec	Character vector of H3 addresses (one per PU; same order as s).
res_vec	Integer vector of H3 resolutions (same length/order as h3_vec).
res_levels	Vector of resolutions in hierarchical order (e.g. c(5, 6, 7)).
nearest_parent_row_of	Integer vector where each position gives the row index of the nearest parent cell in s (or NA if none), as returned by build_h3_maps().
children_by_row	List of integer vectors giving, for each PU row, the row indices of all direct children (finer-resolution descendants), as returned by build_h3_maps().
mode	Either "coarser_first" or "finer_first" (default). Controls whether coarse or fine PUs are retained when duplicates occur.

Details

When mode = "finer_first", removing selected coarser cells can reduce the total selected area if only a subset of their descendant cells were selected (i.e., partial coverage of the parent footprint).

The function operates in a **single pass**, using the precomputed parent/child lists from build_h3_maps(), and produces a new column paste0(sel_col, "_deduplicated").

Value

The input s with an additional column paste0(sel_col, "_deduplicated") containing a clean 0/1 selection.

Examples

```
# One parent (res7) with two children (res8)
parent <- "872a1072bffffff"
kids   <- c("882a1072b1fffff", "882a1072b3fffff")

h3_vec  <- c(parent, kids)
res_vec <- c(7L, 8L, 8L)

maps <- build_h3_maps(h3_vec, res_vec = res_vec)

s <- data.frame(solution_1 = c(1L, 1L, 1L))

# Keep the coarser cell (drops children)
out_coarse <- deduplicate_h3_selections(
  s, sel_col = "solution_1",
```

```
  h3_vec = maps$h3_vec, res_vec = maps$res_vec, res_levels = maps$res_levels,
  nearest_parent_row_of = maps$nearest_parent_row_of,
  children_by_row = maps$children_by_row,
  mode = "coarser_first"
)

# Keep the finer cells (drops parent)
out_fine <- deduplicate_h3_selections(
  s, sel_col = "solution_1",
  h3_vec = maps$h3_vec, res_vec = maps$res_vec, res_levels = maps$res_levels,
  nearest_parent_row_of = maps$nearest_parent_row_of,
  children_by_row = maps$children_by_row,
  mode = "finer_first"
)

out_coarse
out_fine
```

`eval_exact_raster_coverage` *Exact feature coverage from rasters and selected PUs*

Description

Calculate how well raster-based features are represented by a selected set of planning units.

This function evaluates feature coverage from raster layers using exact area-weighted extraction. For each feature, it calculates the total amount available within its native-resolution planning unit footprint, the amount held by the selected planning units, and the proportion of the feature represented by the solution.

If targets are provided, the function also reports absolute and relative shortfalls from those targets.

Usage

```
eval_exact_raster_coverage(
  spp_all,
  pu_sf,
  solution_col,
  targets = NULL,
  res_col = "res"
)
```

Arguments

`spp_all`	A `SpatRaster` or `RasterStack`/`RasterBrick` of feature layers. Layer names must start with `r<res>_` (e.g. `r5_`, `r6_`, `r7_`).
`pu_sf`	Planning units as an `sf` polygon object containing all resolutions. Must include the resolution column specified by `res_col`.

solution_col	Name of the 0/1 (or logical) selection column in pu_sf indicating selected planning units.
targets	Optional named numeric vector of RELATIVE targets (proportions in $[0, 1]$). For example, targets = c(r7_featA = 0.30) means a target of holding 30\ total available amount of r7_featA within its native-resolution domain. If unnamed, the first value is applied to all features.
res_col	Name of the resolution column in pu_sf (default "res").

Value

A data frame with columns:

- feature
- native_res
- total_area Total feature amount within the native-resolution domain (area-integral units).
- held_area Feature amount held by the selected area, restricted to that domain (area-integral units).
- relative_held Relative held amount (held_area / total_area).
- target Relative target proportion (if targets supplied).
- absolute_shortfall Absolute shortfall in feature units ($\max(0, target*total\_area - held\_area)$).
- relative_shortfall Relative shortfall in proportions ($\max(0, target - relative\_held)$).

Examples

```
# Tiny raster with a single layer named with r7_ prefix
r <- terra::rast(nrows = 2, ncols = 2, xmin = 0, xmax = 2, ymin = 0, ymax = 2)
terra::values(r) <- 1
names(r) <- "r7_featA"

# One square PU covering the raster
pu <- sf::st_sf(
  res = 7L,
  selected = 1L,
  geometry = sf::st_sfc(
    sf::st_polygon(list(rbind(c(0, 0), c(2, 0), c(2, 2), c(0, 2), c(0, 0)))),
    crs = 4326
  )
)

eval_exact_raster_coverage(
  spp_all = r,
  pu_sf = pu,
  solution_col = "selected",
  targets = c(r7_featA = 0.3),
  res_col = "res"
)
```

`eval_geom_feature_coverage` *Geometry-based feature coverage evaluation from sf column features and selected PUs*

Description

Calculate how well features are represented by a selected set of planning units.

This function evaluates feature coverage using feature values stored directly in planning unit attributes. For each feature, it calculates the total amount available, the amount held by the selected planning units, and the proportion of the feature represented by the solution.

If targets are provided, the function also reports absolute and relative shortfalls from those targets.

Feature coverage is calculated using fractional overlap between planning units and the union of selected planning units.

Usage

```
eval_geom_feature_coverage(
  s,
  flag_col = "selected_by_resolution",
  feature_cols,
  res_col = "res",
  targets = NULL
)
```

Arguments

`s`	An `sf` object of planning units (all resolutions).
`flag_col`	Name of the 0/1 selection column in `s`.
`feature_cols`	Character vector of feature column names in `s`. Feature names must start with `r<res>_` so the native resolution can be inferred.
`res_col`	Name of the resolution column in `s` (default `"res"`).
`targets`	Optional named numeric vector of RELATIVE targets (proportions in $[0, 1]$). For example, `targets = c(r7_featA = 0.30)` means a target of 30\ available amount of `r7_featA` within its native-resolution footprint. If unnamed, the first value is applied to all features.

Value

A tibble with one row per feature and columns:

- `feature`, `native_res`
- `total` Total feature amount across all PUs at the native resolution.
- `held` Feature amount held by the selected area (via fractional overlap), restricted to the native-resolution footprint.

- `rel_held` Relative held amount (`held / total`).
- `target` Relative target proportion (if `targets` supplied).
- `abs_shortfall` Absolute shortfall in feature units ($\max(0, target * total - held)$).
- `rel_shortfall` Relative shortfall in proportions ($\max(0, target - rel\_held)$).

Examples

```
# Minimal polygons
p1 <- sf::st_polygon(list(rbind(c(0, 0), c(1, 0), c(1, 1), c(0, 1), c(0, 0))))
p2 <- sf::st_polygon(list(rbind(c(1, 0), c(2, 0), c(2, 1), c(1, 1), c(1, 0))))

s <- sf::st_sf(
  res = c(7L, 7L),
  area_km2 = c(1, 1),
  selected_by_resolution = c(1L, 0L),
  r7_featA = c(10, 5),
  geometry = sf::st_sfc(p1, p2, crs = 4326)
)

eval_geom_feature_coverage(
  s,
  flag_col = "selected_by_resolution",
  feature_cols = "r7_featA",
  res_col = "res"
)
```

---

summarize_coverage_by_resolution

Cross-scale coverage summary by resolution

Description

Summarize how much area is selected at each resolution in a multiscale H3 system. This function adjusts for overlap between coarse and fine planning units so that selected area is not double-counted across resolutions.

The output reports, for each resolution, the total available area, the effective selected area (after cross-scale adjustment), and the proportion selected.

Usage

```
summarize_coverage_by_resolution(
  s,
  flag_col = "selected_by_resolution_final",
  cs_idx
)
```

Arguments

s	An sf or data frame of planning units. Must contain columns res and area_km2, plus the selection column referenced by flag_col.
flag_col	Name of the 0/1 selection column to summarise.
cs_idx	A cross-scale index list from build_crossscale_index(), containing at least res_levels, rows_by_res, pos_in_res, anc_at_res, and desc_at_res.

Value

A tibble with columns:

- res – resolution,
- area_total_km2 – total area at that resolution,
- area_selected_km2 – cross-scale selected area, and
- coverage_prop – proportion selected.

Examples

```
parent <- "872a1072bffffff"
kids   <- c("882a1072b1fffff", "882a1072b3fffff")

s <- data.frame(
  h3_address = c(parent, kids),
  res = c(7L, 8L, 8L),
  area_km2 = c(14, 2, 2),
  selected_by_resolution_final = c(0L, 1L, 0L)
)
maps   <- build_h3_maps(s, res_levels = c(7L, 8L))
cs_idx <- build_crossscale_index(maps)

summarize_coverage_by_resolution(s, "selected_by_resolution_final", cs_idx)
```

---

summarize_coverage_by_strata

Cross-scale coverage summary by strata

Description

Summarize how much area is selected within each stratum (e.g. country or region) in a multiscale H3 system. This function reports a cross-scale coverage metric that avoids double-counting when selected planning units overlap across resolutions.

For each stratum, coverage is reported at a single "native" resolution chosen from the planning units labeled with that stratum.

Usage

```
summarize_coverage_by_strata(
  s,
  flag_col = "selected_by_resolution_final",
  cs_idx,
  strata_col = "strata"
)
```

Arguments

s	An sf object or data frame of planning units. Must contain columns res, area_km2, and the strata column referenced by strata_col.
flag_col	Name of the 0/1 selection column to summarize.
cs_idx	A cross-scale index list returned by build_crossscale_index(), containing at least anc_at_res and desc_at_res.
strata_col	Name of the strata column in s (default "strata").

Value

A tibble with one row per stratum and the following columns:

- strata
- native_res
- area_total_native_km2
- area_selected_km2
- coverage_prop

Examples

```
parent <- "872a1072bffffff"
kids   <- c("882a1072b1fffff", "882a1072b3fffff")

s <- data.frame(
  h3_address = c(parent, kids),
  res = c(7L, 8L, 8L),
  strata = c("A", "B", "B"),
  area_km2 = c(14, 2, 2),
  selected_by_resolution_final = c(0L, 1L, 0L)
)
maps   <- build_h3_maps(s, res_levels = c(7L, 8L))
cs_idx <- build_crossscale_index(maps)

summarize_coverage_by_strata(
  s,
  "selected_by_resolution_final",
  cs_idx,
  strata_col = "strata"
)
```

Index