Package 'restoptr'

November 12, 2022

Type Package

Title Ecological Restoration Planning

Version 1.0.3

Description Flexible framework for ecological restoration planning. It aims to identify priority areas for restoration efforts using optimization algorithms (based on Justeau-Allaire et al. 2021 <doi:10.1111/1365-2664.13803>). Priority areas can be identified by maximizing landscape indices, such as the effective mesh size (Jaeger 2000 <doi:10.1023/A:1008129329289>), or the integral index of connectivity (Pascual-Hortal & Saura 2006 <doi:10.1007/s10980-006-0013-z>). Additionally, constraints can be used to ensure that priority areas exhibit particular characteristics (e.g., ensure that particular places are not selected for restoration, ensure that priority areas form a single contiguous network). Furthermore, multiple near-optimal solutions can be generated to explore multiple options in restoration planning. The package leverages the 'Choco-solver' software to perform optimization using constraint programming (CP) techniques (<https://choco-solver.org/>).

License GPL (>= 3)

Encoding UTF-8

Language en-US

URL https://dimitri-justeau.github.io/restoptr/

BugReports https://github.com/dimitri-justeau/restoptr/issues

SystemRequirements Java (>= 11.0.12)

VignetteBuilder knitr

RoxygenNote 7.2.0

Imports utils, assertthat (>= 0.2.1), magrittr, crayon (>= 1.4.1), methods

Depends R (>= 4.1.0), terra (>= 1.5-12), rJava (>= 1.0.5), units (>= 0.8-0)

Suggests testthat (>= 2.0.1), knitr (>= 1.2.0), roxygen2 (>= 6.1.1), rmarkdown (>= 1.10), landscapemetrics (>= 1.5.4), raster (>= 3.5), rgdal, vegan (>= 2.5.7), cluster (>= 2.1.2), ggthemes, paletteer

Collate 'internal.R' 'add_available_areas_constraint.R'
'add_compactness_constraint.R' 'add_components_constraint.R'
'add_connected_constraint.R' 'add_locked_out_constraint.R'
'add_min_iic_constraint.R' 'add_min_mesh_constraint.R'
'add_restorable_constraint.R' 'add_settings.R'
'is_java_available.R' 'package.R' 'preprocessing.R'
'restopt_component.R' 'restopt_problem.R' 'restopt_solution.R'
'set_max_iic_objective.R' 'set_max_mesh_objective.R'
'set_max_nb_pus_objective.R' 'set_max_restore_objective.R'
'set_min_nb_pus_objective.R' 'set_min_restore_objective.R'
'set_no_objective.R' 'solve.R' 'terra_io.R' 'terra_utils.R'
'utils-pipe.R' 'zzz.R'

NeedsCompilation no

Author Dimitri Justeau-Allaire [aut, cre]
(<https://orcid.org/0000-0003-4129-0764>),
Jeffrey O Hanson [aut] (<https://orcid.org/0000-0002-4716-6134>),
Ghislain Vieilledent [aut] (<https://orcid.org/0000-0002-1685-4997>),
Philippe Vismara [aut],
Xavier Lorca [aut],
Philippe Birnbaum [aut]

Maintainer Dimitri Justeau-Allaire <dimitri.justeau@gmail.com>

Repository CRAN

Date/Publication 2022-11-12 14:00:02 UTC

R topics documented:

`add_available_areas_constraint` *Add available areas constraint*

Description

Add constraint to a restoration problem (`restopt_problem()`) object to specify that only certain planning units can be selected for restoration activities.

Usage

```
add_available_areas_constraint(problem, data, touches = FALSE)
```

Arguments

`problem`	`restopt_problem()` Restoration problem object.
`data`	`terra::rast()` or `terra::vect()` Either a raster object containing binary values hat indicate which planning units can be selected for restoration (i.e., only cells with a value equal one are available), or a vector object whose features correspond to the available areas.
`touches`	`logical` If the available area data is a vector, define wether the rasterization must include all pixels touching the polygons. (see `terra::rasterize()`). Useless if the data is raster data.

Details

Available areas constraints can be used to incorporate a wide range of criteria into restoration planning problems. They can be used to account for existing land-use practices, feasibility of restoration activities, and stakeholder preferences. For example, available areas constraints can be used to ensure that urban areas are not selected for restoration. Additionally, if restoration activities can only be implemented depending on certain conditions – such as places where landscape slope is not too steep – then available areas constraints could be used to ensure restoration activities are not prioritized for places where they could not be implemented. Furthermore, if stakeholders require solutions that do not prioritize particular places for restoration, then available areas constraints can also be used to achieve this. See `add_locked_out_constraint()`, which achieve the same as available areas constraint by defining areas that are NOT available for restoration. **Note**: The locked out constraint and the available are the same constraints, with a different input data. Thus, from a modelling perspective, `add_available_areas_constraint()` is just a pre processing layer in front of `add_locked_out_constraint()`. This is why if you print a restopt problem with an available areas constraint, you will see a locked out constraint.

Value

An updated restoration problem (`restopt_problem()`) object.

See Also

Other constraints: `add_compactness_constraint()`, `add_components_constraint()`, `add_connected_constraint()`, `add_locked_out_constraint()`, `add_min_iic_constraint()`, `add_min_mesh_constraint()`, `add_restorable_constraint()`

Examples

```
# load data
habitat_data <- rast(
  system.file("extdata", "habitat_hi_res.tif", package = "restoptr")
)

available <- vect(
system.file("extdata", "accessible_areas.gpkg", package = "restoptr")
)

# plot data
plot(habitat_data)
plot(available, add=TRUE)

# create problem with available areas constraints
p <- restopt_problem(
    existing_habitat = habitat_data,
    aggregation_factor = 16,
    habitat_threshold = 0.7
  ) %>%
  set_max_iic_objective() %>%
  add_restorable_constraint(
```

```
    min_restore = 5,
    max_restore = 5,
  ) %>%
  add_available_areas_constraint(available) %>%
  add_settings(time_limit = 1)

# print problem
print(p)

# solve problem
s <- solve(p)

# plot solution
plot(s)
```

`add_compactness_constraint` *Add constraint to limit compactness*

Description

Add constraint to a restoration problem (`restopt_problem()`) object to specify the compactness of a solution.

Usage

```
add_compactness_constraint(problem, max_diameter, unit = "m")
```

Arguments

`problem`	`restopt_problem()` Restoration problem object.
`max_diameter`	`numeric` Maximum diameter value.
`unit`	`unit` object or a `character` that can be coerced to an area unit (see `unit` package), or "cells" for cell width of aggregated habitat raster. Corresponds to the unit of the maximum diameter. If the input habitat raster does not use a projected coordinate system, only "cells" is available.

Details

The compactness constraint is defined according to the diameter of the smallest enclosing circle which contains the center of selected planning units for restoration (see https://w.wiki/4vfg). The unit of the diameter corresponds either to a unit available in the `unit` package, or to planning unit width ("cells"). Note that, as the computation occurs on aggregated cells, if `max_diameter` is used with a different unit than "cells", it will be rounded to the closest corresponding number of cells. For example, a diameter of 4 cells means that no more than 4 cells can be found in line in the solution. In practice, this constraint is useful to ensure the feasibility of a restoration project, and to

integrate economies of scale. Compact restoration areas are usually associated with lower costs and easier management, because it ensures that restoration sites are not too far away from each other (e.g. lower travel costs between sites, less areas to monitor, etc.).

Value

An updated restoration problem (`restopt_problem()`) object.

See Also

Other constraints: `add_available_areas_constraint()`, `add_components_constraint()`, `add_connected_constraint`
`add_locked_out_constraint()`, `add_min_iic_constraint()`, `add_min_mesh_constraint()`,
`add_restorable_constraint()`

Examples

```
# load data
habitat_data <- rast(
  system.file("extdata", "habitat_hi_res.tif", package = "restoptr")
)

# create problem
p <- restopt_problem(
    existing_habitat = habitat_data,
    aggregation_factor = 16,
    habitat_threshold = 0.7
  ) %>%
  add_restorable_constraint(
    min_restore = 200,
    max_restore = 300,
  ) %>%
  add_compactness_constraint(1800, unit = "m")

# plot preprocessed data
plot(rast(list(p$data$existing_habitat, p$data$restorable_habitat)), nc = 2)

# print problem
print(p)

# Solve problem
s <- solve(p)
# plot solution
plot(s)
```

add_components_constraint *Add constraint to limit the number of connected components*

Description

Add constraint to a restoration problem (restopt_problem()) object to specify the number of connected components that can be present within a solution.

Usage

```
add_components_constraint(problem, min_nb_components, max_nb_components)
```

Arguments

problem	restopt_problem() Restoration problem object.
min_nb_components	integer Minimum number of connected components.
max_nb_components	integer Maximum number of connected components.

Details

A connected component is a spatially continuous set of planning units. This constraints applies on the set of planning units that are selected for restoration, and allows to specify a minimum and maximum number of connected components. In practice, this constraint is useful to ensure the feasibility of a restoration project, and to integrate economies of scale. Continuous restoration areas (i.e. less connected components) are usually associated with lower costs, because it ensures that restoration sites are not too far away from each other (e.g. lower travel costs between sites, less areas to monitor, etc.). On the other hand, it can be useful to enforce several disconnected restoration areas to ensure that hazards (e.g. fire) do not strike all planning units at the same time.

Value

An updated restoration problem (restopt_problem()) object.

See Also

Other constraints: add_available_areas_constraint(), add_compactness_constraint(), add_connected_constrair
add_locked_out_constraint(), add_min_iic_constraint(), add_min_mesh_constraint(),
add_restorable_constraint()

Examples

```
# load data
habitat_data <- rast(
  system.file("extdata", "habitat_hi_res.tif", package = "restoptr")
)

# create problem
p <- restopt_problem(
    existing_habitat = habitat_data,
    aggregation_factor = 16,
    habitat_threshold = 0.7
  ) %>%
  add_restorable_constraint(
    min_restore = 10,
    max_restore = 100,
  ) %>%
  add_components_constraint(1, 1)

# plot preprocessed data
plot(rast(list(p$data$existing_habitat, p$data$restorable_habitat)), nc = 2)

# print problem
print(p)

# Solve problem
s <- solve(p)
# plot solution
plot(s)
```

`add_connected_constraint` *Add constraint to ensure that the selected planning units for restoration are connected.*

Description

Add constraint to a restoration problem (`restopt_problem()`) object to specify the selected planning units are connected

Usage

```
add_connected_constraint(problem)
```

Arguments

`problem`	`restopt_problem()` Restoration problem object.

Details

A connected area is such that there is a path between any to planning units within the area. This constraints applies on the set of planning units that are selected for restoration. In practice, this constraint is useful to ensure the feasibility of a restoration project, and to integrate economies of scale. Connected restoration areas are usually associated with lower costs, because it ensures that restoration sites are not too far away from each other (e.g. lower travel costs between sites, less areas to monitor, etc.). **Note** This constraint relies on the add_components_constraint(), with parameters set to enforce exactly one connected component. Also see add_components_constraint and add_compactness_constraint.

Value

An updated restoration problem (`restopt_problem()`) object.

See Also

Other constraints: `add_available_areas_constraint()`, `add_compactness_constraint()`, `add_components_constrai`
`add_locked_out_constraint()`, `add_min_iic_constraint()`, `add_min_mesh_constraint()`,
`add_restorable_constraint()`

Examples

```
# load data
habitat_data <- rast(
  system.file("extdata", "habitat_hi_res.tif", package = "restoptr")
)

# create problem
p <- restopt_problem(
    existing_habitat = habitat_data,
    aggregation_factor = 16,
    habitat_threshold = 0.7
  ) %>%
  add_restorable_constraint(
    min_restore = 10,
    max_restore = 100,
  ) %>%
  add_connected_constraint()

# plot preprocessed data
plot(rast(list(p$data$existing_habitat, p$data$restorable_habitat)), nc = 2)

# print problem
print(p)

# Solve problem
s <- solve(p)
# plot solution
plot(s)
```

`add_locked_out_constraint` *Add locked out constraint*

Description

Add constraint to a restoration problem (`restopt_problem()`) object to specify that certain planning units cannot be selected for restoration activities.

Usage

```
add_locked_out_constraint(problem, data, touches = FALSE)
```

Arguments

`problem`	`restopt_problem()` Restoration problem object.
`data`	`terra::rast()` or `terra::vect()` Either a raster object containing binary values hat indicate which planning units cannot be selected for any restoration (i.e., cells with a value equal one are locked out from the solution), or a vector object whose features correspond to the locked out areas. See the function `add_available_areas_constraint()` to get a locked out constraint from allowed restoration areas.
`touches`	`logical` If the locked out data is a vector, define wether the rasterization must include all pixels touching the polygons. (see `terra::rasterize()`). Useless if the data is raster data.

Details

Locked out constraints can be used to incorporate a wide range of criteria into restoration planning problems. They can be used to account for existing land-use practices, feasibility of restoration activities, and stakeholder preferences. For example, locked out constraints can be used to ensure that urban areas are not selected for restoration. Additionally, if restoration activities can only be implemented depending on certain conditions – such as places where landscape slope is not too steep – then locked out constraints could be used to ensure restoration activities are not prioritized for places where they could not be implemented. Furthermore, if stakeholders require solutions that do not prioritize particular places for restoration, then locked out constraints can also be used to achieve this. See `add_available_areas_constraint()`, which achieve the same as the locked out constraint by defining areas that ARE available for restoration.

Value

An updated restoration problem (`restopt_problem()`) object.

See Also

Other constraints: add_available_areas_constraint(), add_compactness_constraint(), add_components_constrai
add_connected_constraint(), add_min_iic_constraint(), add_min_mesh_constraint(), add_restorable_constra

Examples

```
# load data
habitat_data <- rast(
  system.file("extdata", "habitat_hi_res.tif", package = "restoptr")
)

locked_out_data <- rast(
 system.file("extdata", "locked_out.tif", package = "restoptr")
)

# plot data
plot(rast(list(habitat_data, locked_out_data)), nc = 2)

# create problem with locked out constraints
p <- restopt_problem(
    existing_habitat = habitat_data,
    aggregation_factor = 16,
    habitat_threshold = 0.7
  ) %>%
  set_max_iic_objective() %>%
  add_restorable_constraint(
    min_restore = 5,
    max_restore = 5,
  ) %>%
  add_locked_out_constraint(data = locked_out_data) %>%
  add_settings(time_limit = 1)

# print problem
print(p)

# solve problem
s <- solve(p)

# plot solution
plot(s)
```

`add_min_iic_constraint` *Add constraint to enforce a minimum integral index of connectivity (IIC) value*

Description

Add constraint to a restoration problem (`restopt_problem()`) object to specify the minimum integral index of connectivity of a solution.

Usage

```
add_min_iic_constraint(
  problem,
  min_iic,
  distance_threshold = -1,
  unit = "m",
  precision = 4
)
```

Arguments

`problem`	`restopt_problem()` Restoration problem object.
`min_iic`	`numeric` Minimum IIC value (between 0 and 1).
`distance_threshold`	`numeric` greater than 0. Minimum distance (in `unit`) between two patches to consider them connected in the computation of the IIC. The default value -1 causes the function to use 1 aggregated cell as the distance threshold.
`unit`	`unit` object or a `character` that can be coerced to a distance unit (see `unit` package), or "cells" for cell width of aggregated habitat raster. Units of the `distance_threshold` parameter. If the input habitat raster does not use a projected coordinate system, only "cells" is available. Meters by default, expected if `distance_threshold` is set to its default value (-1), which causes the function to use 1 cell by default.
`precision`	`integer` Precision for calculations. Defaults to 4.

Details

The integral index of connectivity (IIC) is a graph-based inter-patch connectivity index based on a binary connection model (Pascual-Hortal & Saura, 2006). Its maximization in the context of restoration favours restoring the structural connectivity between large patches. IIC is unitless and comprised between 0 (no connectivity) and 1 (all the landscape is habitat, thus fully connected). The `distance_threshold` parameter indicates to the solver how to construct the habitat graph, i.e. what is the minimum distance between two patches to consider them as connected. Note that, as the computation occurs on aggregated cells, if `distance_threshold` is used with a different unit than "cells", it will be rounded to the closest corresponding number of cells.

The effective mesh size (MESH) is a measure of landscape fragmentation based on the probability that two randomly chosen points are located in the same patch (Jaeger, 2000). Maximizing it in the context of restoration favours fewer and larger patches.

Value

An updated restoration problem (`restopt_problem()`) object.

References

Pascual-Hortal, L., & Saura, S. (2006). Comparison and development of new graph-based landscape connectivity indices: Towards the priorization of habitat patches and corridors for conservation. Landscape Ecology, 21(7), 959-967. https://doi.org/10.1007/s10980-006-0013-z

See Also

set_max_iic_objective

Other constraints: add_available_areas_constraint(), add_compactness_constraint(), add_components_constrai
add_connected_constraint(), add_locked_out_constraint(), add_min_mesh_constraint(),
add_restorable_constraint()

Examples

```
# load data
habitat_data <- rast(
  system.file("extdata", "habitat_hi_res.tif", package = "restoptr")
)

locked_out_data <- rast(
 system.file("extdata", "locked_out.tif", package = "restoptr")
)

# create problem with locked out constraints
p <- restopt_problem(
    existing_habitat = habitat_data,
    aggregation_factor = 16,
    habitat_threshold = 0.7
  ) %>%
  add_min_iic_constraint(0.2) %>%
  add_restorable_constraint(
    min_restore = 200,
    max_restore = 300,
  ) %>%
  add_locked_out_constraint(data = locked_out_data) %>%
  add_compactness_constraint(2500, unit = "m") %>%
  add_settings(time_limit = 1)

# print problem
print(p)

# solve problem
s <- solve(p)

# plot solution
plot(s)
```

`add_min_mesh_constraint` *Add constraint to enforce a minimum effective mesh size (MESH) value*

Description

Add constraint to a restoration problem (`restopt_problem()`) object to specify the minimum effective mesh size of a solution.

Usage

```
add_min_mesh_constraint(problem, min_mesh, precision = 4, unit = "ha")
```

Arguments

`problem`	`restopt_problem()` Restoration problem object.
`min_mesh`	`numeric` Minimum MESH value.
`precision`	`integer` Precision for calculations. Defaults to 4.
`unit`	`unit` object or a `character` that can be coerced to an area unit (see `unit` package), or "cells" for cell width of aggregated habitat raster. Corresponds to the unit of the minimum mesh value If the input habitat raster does not use a projected coordinate system, only "cells" is available. Defaults to "ha".

Details

The effective mesh size (MESH) is a measure of landscape fragmentation based on the probability that two randomly chosen points are located in the same patch (Jaeger, 2000). Maximizing it in the context of restoration favours fewer and larger patches.

Value

An updated restoration problem (`restopt_problem()`) object.

References

Jaeger, J. A. G. (2000). Landscape division, splitting index, and effective mesh size: New measures of landscape fragmentation. Landscape Ecology, 15(2), 115-130. https://doi.org/10.1023/A:1008129329289

See Also

set_max_mesh_objective

Other constraints: `add_available_areas_constraint()`, `add_compactness_constraint()`, `add_components_constrai`
`add_connected_constraint()`, `add_locked_out_constraint()`, `add_min_iic_constraint()`,
`add_restorable_constraint()`

Examples

```
# load data
habitat_data <- rast(
  system.file("extdata", "habitat_hi_res.tif", package = "restoptr")
)

# create problem
p <- restopt_problem(
    existing_habitat = habitat_data,
    aggregation_factor = 16,
    habitat_threshold = 0.7
  ) %>%
  add_restorable_constraint(
    min_restore = 200,
    max_restore = 300,
  ) %>%
  add_min_mesh_constraint(min_mesh = 2500, unit = "ha")

# plot preprocessed data
plot(rast(list(p$data$existing_habitat, p$data$restorable_habitat)), nc = 2)

# print problem
print(p)

# Solve problem
s <- solve(p)
# plot solution
plot(s)
```

`add_restorable_constraint` *Add constraint to specify the available amount of surface for restoration*

Description

Add constraint to a restoration problem (`restopt_problem()`) object to specify specify the available amount of surface for restoration

Usage

```
add_restorable_constraint(
  problem,
  min_restore,
  max_restore,
  min_proportion = 1,
```

```
  unit = "ha"
)
```

Arguments

problem	restopt_problem() Restoration problem object.
min_restore	numeric Minimum allowed area to restore in the solution.
max_restore	numeric Maximum allowed area to restore in the solution
min_proportion	float Minimum habitat proportion to consider a cell as restored.
unit	unit object or a character that can be coerced to an area unit (see unit package), or "cells" for number of cells from the original habitat raster). If the input habitat raster does not use a projected coordinate system, only "cells" is available.

Details

Given the restorable_habitat input raster in restopt_problem, this constraint ensures that the total amount of restorable habitat in selected planning units is at least min_restore and at most max_restore. The unit of min_restore and max_restore can be either in a surface unit handled by the unit package, or in number of cells from the original habitat input raster ("cells"). The min_proportion parameter is a numeric between 0 and 1, and correspond to the minimum proportion of habitat area that needs to be restored in the planning unit to consider the planning unit as restored. This proportion is relative to the area of a planning unit, which is computed automatically from the input habitat raster. Note that planning unit area is considered uniform, and the distortion is not corrected. It could be using the cellSize function of the terra package, but this function is currently pretty slow for large rasters. If your problem is at regional scale, the distortion should be negligible. However, at larger scales, the best is to use an equal-area projected coordinate system.

Note that when a solution is found, the "maximum restorable habitat" is displayed, this value does not correspond to the max_restore parameter, but to the total area that can be restored in the selected planning units. The max_restore parameter is actually an upper bound of the minimum habitat that needs to be restored to reach the min_proportion of habitat in every selected planning units.

Value

An updated restoration problem (restopt_problem()) object.

See Also

Other constraints: add_available_areas_constraint(), add_compactness_constraint(), add_components_constrai
add_connected_constraint(), add_locked_out_constraint(), add_min_iic_constraint(),
add_min_mesh_constraint()

Examples

```
# load data
habitat_data <- rast(
```

```
  system.file("extdata", "habitat_hi_res.tif", package = "restoptr")
)

# create problem
p <- restopt_problem(
    existing_habitat = habitat_data,
    aggregation_factor = 16,
    habitat_threshold = 0.7
  ) %>%
  add_restorable_constraint(
    min_restore = 200,
    max_restore = 300,
    min_proportion = 0.7
  ) %>%
  add_compactness_constraint(5, unit = "cells")

# print problem
print(p)

# plot preprocessed data
plot(rast(list(get_existing_habitat(p),
               get_restorable_habitat(p),
               get_locked_out_areas(p))), nc = 3)

# Solve problem
s <- solve(p)
# plot solution
plot(s)
```

---

add_settings *Add settings*

Description

Add settings to a restoration problem (restopt_problem()) object to customize the optimization procedure.

Usage

```
add_settings(
  problem,
  precision = 4,
  time_limit = 0,
  nb_solutions = 1,
  optimality_gap = 0,
  solution_name_prefix = "Solution "
)
```

Arguments

problem	restopt_problem() Restoration problem object.
precision	integer Precision for calculations. Defaults to 4.
time_limit	integer Maximum permitted run time for optimization (seconds). Defaults to 0.
nb_solutions	integer Number of desired solutions. Defaults to 1.
optimality_gap	numeric Optimality gap (between 0 and 1). For example, an argument of 0.1 means that solutions should be within 10% of optimality. Defaults to 0, such that optimal solutions are returned.
solution_name_prefix	character Prefix for the name of solutions. Defaults to "Solution "

Value

An updated restoration problem (restopt_problem()) object.

Examples

```
# load data
habitat_data <- rast(
  system.file("extdata", "habitat_hi_res.tif", package = "restoptr")
)

# create problem
p <- restopt_problem(
    existing_habitat = habitat_data,
    aggregation_factor = 16,
    habitat_threshold = 0.7
  ) %>%
  add_settings(time_limit = 1, precision = 4, nb_solutions = 2)
# print problem
print(p)
```

---

get_aggregation_factor

Retrieve the aggregation factor of a restopt problem.

Description

Retrieve the aggregation factor of a restopt problem.

Usage

```
get_aggregation_factor(problem)
```

Arguments

problem restopt_problem() Restoration problem object.

Value

numeric The aggregation factor of the restopt problem.

Examples

```
#' # load data
habitat_data <- rast(
  system.file("extdata", "habitat_hi_res.tif", package = "restoptr")
)

# create problem
problem <- restopt_problem(
       existing_habitat = habitat_data,
       aggregation_factor = 4,
       habitat_threshold = 0.7
)

get_aggregation_factor(problem)
```

---

get_cell_area *Retrieve the aggregated cell area of a restopt problem.*

Description

Retrieve the aggregated cell area of a restopt problem.

Usage

```
get_cell_area(problem)
```

Arguments

problem restopt_problem() Restoration problem object.

Value

terra::rast() The aggregated cell area of the restopt problem.

Examples

```
#' # load data
habitat_data <- rast(
  system.file("extdata", "habitat_hi_res.tif", package = "restoptr")
)

# create problem
problem <- restopt_problem(
       existing_habitat = habitat_data,
       aggregation_factor = 4,
       habitat_threshold = 0.7
)

get_cell_area(problem)
```

`get_constraints` *Retrieve the constraints of a restopt problem.*

Description

Retrieve the constraints of a restopt problem.

Usage

```
get_constraints(problem)
```

Arguments

`problem`	`restopt_problem()` Restoration problem object.

Value

`list` The constraints of the restopt problem.

Examples

```
#' # load data
habitat_data <- rast(
  system.file("extdata", "habitat_hi_res.tif", package = "restoptr")
)

# create problem
problem <- restopt_problem(
       existing_habitat = habitat_data,
       aggregation_factor = 4,
```

```
       habitat_threshold = 0.7
)

get_constraints(problem)
```

---

get_existing_habitat *Retrieve the existing (i.e. aggregated) habitat data.*

Description

Retrieve the existing (i.e. aggregated) habitat data.

Usage

```
get_existing_habitat(problem)
```

Arguments

problem	restopt_problem()	Restoration problem object.

Value

terra::rast() The existing (aggregated) habitat data.

Examples

```
#' # load data
habitat_data <- rast(
  system.file("extdata", "habitat_hi_res.tif", package = "restoptr")
)

# create problem
problem <- restopt_problem(
       existing_habitat = habitat_data,
       aggregation_factor = 4,
       habitat_threshold = 0.7
)

plot(get_existing_habitat(problem))
```

get_habitat_threshold *Retrieve the habitat threshold parameter of a restopt problem.*

Description

Retrieve the habitat threshold parameter of a restopt problem.

Usage

```
get_habitat_threshold(problem)
```

Arguments

problem	restopt_problem()	Restoration problem object.

Value

numeric The habitat threshold parameter of the restopt problem.

Examples

```
#' # load data
habitat_data <- rast(
  system.file("extdata", "habitat_hi_res.tif", package = "restoptr")
)

# create problem
problem <- restopt_problem(
       existing_habitat = habitat_data,
       aggregation_factor = 4,
       habitat_threshold = 0.7
)

get_habitat_threshold(problem)
```

get_locked_out_areas *Retrieve the locked out areas of a restopt problem.*

Description

Retrieve the locked out areas of a restopt problem.

Usage

```
get_locked_out_areas(problem)
```

Arguments

problem restopt_problem() Restoration problem object.

Value

terra::rast() The locked out areas of the restopt problem.

Examples

```
#' # load data
habitat_data <- rast(
  system.file("extdata", "habitat_hi_res.tif", package = "restoptr")
)

# create problem
problem <- restopt_problem(
       existing_habitat = habitat_data,
       aggregation_factor = 4,
       habitat_threshold = 0.7
)

get_locked_out_areas(problem)
```

---

get_metadata *Restopt solution metadata*

Description

Return the metadata associated with a restopt solution, which contains the characteristics of the solution. The unit for area characteristics can be chosen among "ha" (hectares), "m" (square meters), "km" (square kilometers), and "cells" (cells from the original input habitat raster). Note that the solving time is expressed in seconds.

Usage

```
get_metadata(restopt_solution, area_unit = "ha", distance_unit = "m")
```

Arguments

restopt_solution
: restopt_solution() Restopt solution to this solution.

area_unit
: unit object or a character that can be coerced to an area unit (see unit package), or "cells" for number of cells from the original habitat raster). Unit for areas. If the input habitat raster does not use a projected coordinate system, only "cells" is available. Default is "ha"

distance_unit unit object or a character that can be coerced to an area unit (see unit package), or "cells" for number of cell width from the original habitat raster). Unit for distances. If the input habitat raster does not use a projected coordinate system, only "cells" is available. Default is "m"

Value

A list containing the characteristics of the restopt solution.

---

get_objective *Retrieve the optimization objective of a restopt problem.*

Description

Retrieve the optimization objective of a restopt problem.

Usage

```
get_objective(problem)
```

Arguments

problem restopt_problem() Restoration problem object.

Value

RestoptObjectve The optimization objective of the restopt problem.

Examples

```
#' # load data
habitat_data <- rast(
  system.file("extdata", "habitat_hi_res.tif", package = "restoptr")
)

# create problem
problem <- restopt_problem(
       existing_habitat = habitat_data,
       aggregation_factor = 4,
       habitat_threshold = 0.7
)

get_objective(problem)
```

get_original_habitat *Retrieve the original (i.e. not aggregated) habitat data.*

Description

Retrieve the original (i.e. not aggregated) habitat data.

Usage

```
get_original_habitat(problem)
```

Arguments

problem	restopt_problem()	Restoration problem object.

Value

terra::rast() The original (i.e. not aggregated) habitat data.

Examples

```
#' # load data
habitat_data <- rast(
  system.file("extdata", "habitat_hi_res.tif", package = "restoptr")
)

# create problem
problem <- restopt_problem(
       existing_habitat = habitat_data,
       aggregation_factor = 4,
       habitat_threshold = 0.7
)

plot(get_original_habitat(problem))
```

get_restorable_habitat *Retrieve the restorable habitat (aggregated) data.*

Description

Retrieve the restorable habitat (aggregated) data.

Usage

```
get_restorable_habitat(problem)
```

Arguments

problem	restopt_problem() Restoration problem object.

Value

terra::rast() The restorable habitat (aggregated) data.

Examples

```
#' # load data
habitat_data <- rast(
  system.file("extdata", "habitat_hi_res.tif", package = "restoptr")
)

# create problem
problem <- restopt_problem(
       existing_habitat = habitat_data,
       aggregation_factor = 4,
       habitat_threshold = 0.7
)

plot(get_restorable_habitat(problem))
```

---

get_settings *Retrieve the settings of a restopt problem.*

Description

Retrieve the settings of a restopt problem.

Usage

```
get_settings(problem)
```

Arguments

problem	restopt_problem() Restoration problem object.

Value

list The settings associated with the restopt problem.

Examples

```
#' # load data
habitat_data <- rast(
  system.file("extdata", "habitat_hi_res.tif", package = "restoptr")
)

# create problem
problem <- restopt_problem(
       existing_habitat = habitat_data,
       aggregation_factor = 4,
       habitat_threshold = 0.7
)

get_settings(problem)
```

`invert_vector` *Invert a vector layer according to the extent of a restopt problem.*

Description

Invert a vector layer according to the extent of a restopt problem.

Usage

```
invert_vector(vector_layer, extent = NULL, filter = NULL)
```

Arguments

`vector_layer`	`terra::vect()` Vector layer.
`extent`	`SpatExtent` Optional: you can specify another extent as the input vector layer extent for the inversion.
`filter`	Optional: filter to apply to `x`. Leave NULL for no filtering.

Details

Invert a vector layer according to its extent, or a user-specified extent. This function is useful to derive locked out areas from accessible areas, e.g. buffer around tracks.

Value

A `terra::vect()` Vector object.

Examples

```
habitat_data <- rast(
  system.file("extdata", "habitat_hi_res.tif", package = "restoptr")
)
available <- vect(
  system.file("extdata", "accessible_areas.gpkg", package = "restoptr")
)
locked_out <- invert_vector(
  vector_layer = available,
  extent = ext(habitat_data),
  filter = available$ID==2
)
```

---

is_java_available *Is Java is available?*

Description

The **restoptr** package uses Java to perform optimization procedures. As such, it is important that Java is installed correctly when using this package. This function verifies if Java is available on the system.

Usage

```
is_java_available()
```

Value

A `logical` (`TRUE` or `FALSE`) value indicating if Java is available for usage.

---

preprocess_input *Restopr input preprocessing function.*

Description

From a binary, possibly high resolution, habitat raster, this function produces three input rasters for restopt:

Usage

```
preprocess_input(habitat, habitat_threshold = 1, aggregation_factor = 1)
```

Arguments

habitat terra::rast() Raster object containing binary values that indicate if each planning unit contains habitat or not. Cells with the value 1 must correspond to existing habitat. Cells with the value 0 must correspond to degraded (or simply non-habitat) areas. Finally, NA (or NO_DATA) cells are considered to be outside of the landscape.

habitat_threshold
numeric Number between 0 and 1 indicating, when the habitat raster is downsampled, the minimum proportion of habitat cells (from the original raster) are required within the downsampled raster to be considered as habitat.

aggregation_factor
integer positive integer corresponding to the level of downsampling that will be applied to the habitat. This parameter is important to ensure the tractability of a problem.

Details

- The binary habitat raster (existing_habitat), which can be the same as the input (aggregation factor = 1), but most often is a downsampled version of it, to ensure the tractability of the problem.
- The restorable habitat raster (restorable_habitat), which is a raster indicating how much habitat can be restored. If the aggregation factor = 1, the the restorable habitat raster is binary and the inverse of the habitat raster. Else, the surface of habitat is computed according to the spatial resolution, and the number of habitat pixel present in one larger aggregated cell.
- The cell area raster (cell_area), which correspond for each aggregated cell to the number of number of cells in the input raster. This is necessary because the cell area can be less than expected if the aggregated cell lies in the boundary of study area.

This preprocessing function produces the necessary inputs of a restopt problem from a single binary habitat raster (habitat), which can be at high resolution. Restopt solves a hard constrained combinatorial problem (it can be reduced to a constrained 0/1 knapsack problem, which is know to be NP-Complete), thus the input resolution might need to be reduced to ensure a tractable problem. Performing this downsampling in a systematic and reproducible way is the aim of this function, which relies on the terra::aggregate() function to do it. The aggregation_factor parameter indicates how much the resolution must be reduced. An aggregated pixel will contain at most aggregation_factor^2 pixels from the input habitat raster (cell_area raster in this function outputs). If an aggregated pixel is close to the spatial boundaries of the problem (i.e. NA cells), it can contain less than aggregation_factor^2 fine grained pixels. The habitat_threshold parameter indicates the minimum proportion of habitat pixels (relative to cell_area) whose value is 1 to consider an aggregated pixel as habitat (downsampled_habitat output raster). The restorable_area output raster correspond to the number of pixel with value 0 in aggregated pixels.

Value

A list : list(existing_habitat=downsampled_habitat, restorable_habitat=restorable_habitat, cell_area=cell_area)

Examples

```
# load data
habitat_data <- rast(
  system.file("extdata", "habitat_hi_res.tif", package = "restoptr"))
data <- preprocess_input(
    habitat = habitat_data,
    habitat_threshold = 0.7,
    aggregation_factor = 16
)
```

`print.RestoptProblem` *Print a restoration optimization problem*

Description

Display information about a restoration optimization problem.

Usage

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

Arguments

x	`restopt_problem()` Restoration problem object.
...	Arguments not used.

Examples

```
#' # load data
habitat_data <- rast(
  system.file("extdata", "habitat_hi_res.tif", package = "restoptr")
)

# create problem
p <- restopt_problem(
       existing_habitat = habitat_data,
       aggregation_factor = 4,
       habitat_threshold = 0.7
)

# print problem
print(p)
```

restoptr *restoptr: Ecological Restoration Planning*

Description

The `restoptr` package relies on Constraint Programming (CP) to build and solve ecological restoration planning problems. A `restoptr` problem starts from an existing habitat (raster), where the aim is to identify optimal areas that are suitable for restoration. Several constraints are available to define what is expected for a suitable area for (e.g. it must be connected, compact, must respect a budget, etc). Several optimization objective are also available to define what is a good restoration area (e.g. it must reduce fragmentation, increase ecological connectivity, minimize costs, etc.). `restoptr` relies on advanced landscape indices such as the effective mesh size (Jaeger, 2000), or the integral index of connectivity (Pascual-Hortal & Saura, 2006) to address complex restoration planning problems.

Details

`restoptr` relies on Choco-solver (https://choco-solver.org/), an open-source Java CP solver (Prud'homme et al., 2017). The computationally intensive solving part is thus delegated to Java (see `restopt`, https://github.com/dimitri-justeau/restopt), and the communication between R and Java is handled with the `rJava` package. Therefore, a Java Runtime Environment (>= 8) is necessary to use `restopt`.

Note that the methodology used in `restoptr` was first described in Justeau-Allaire et al. (2021), but `restoptr` provides much more flexibility, new features (e.g. reliable and consistent data preprocessing), new constraints, new optimization objectives, and an improved computational efficiency. Also note that the API was inspired by the `prioritizr` package.

This package contains several vignettes to detail its usage and showcase its features. You can explore these vignettes using `browseVignettes("restoptr")`

References

Hanson JO, Schuster R, Morrell N, Strimas-Mackey M, Edwards BPM, Watts ME, Arcese P, Bennett J, Possingham HP (2022). prioritizr: Systematic Conservation Prioritization in R. R package version 7.1.1. Available at https://CRAN.R-project.org/package=prioritizr.

Jaeger, J. A. G. (2000). Landscape division, splitting index, and effective mesh size: New measures of landscape fragmentation. Landscape Ecology, 15(2), 115-130.

Justeau-Allaire, D., Vieilledent, G., Rinck, N., Vismara, P., Lorca, X., & Birnbaum, P. (2021). Constrained optimization of landscape indices in conservation planning to support ecological restoration in New Caledonia. Journal of Applied Ecology, 58(4), 744-754.

Pascual-Hortal, L., & Saura, S. (2006). Comparison and development of new graph-based landscape connectivity indices: Towards the priorization of habitat patches and corridors for conservation. Landscape Ecology, 21(7), 959-967.

Prud'homme, C., Fages, J.-G., & Lorca, X. (2017). Choco documentation.

`restopt_problem` *Restoration optimization problem*

Description

Create a new restoration optimization problem (`RestoptProblem`) using data that describe the spatial distribution of existing habitat (potentially at high resolution), and parameters to derive a downsampled existing habitat raster, suitable for a tractable optimization, and a restorable habitat raster. Constraints can be added to a restopt problem using `add_****_constraint()` functions, and an optimization objective can be set using `set_****_objective()` functions.

Usage

```
restopt_problem(
  existing_habitat,
  habitat_threshold = 1,
  aggregation_factor = 1
)
```

Arguments

`existing_habitat`
: `terra::rast()` Raster object containing binary values that indicate if each planning unit contains habitat or not. Cells with the value `1` must correspond to existing habitat. Cells with the value `0` must correspond to degraded (or simply non-habitat) areas. Finally, `NA` (or `NO_DATA`) cells are considered to be outside of the landscape. This raster can have a high resolution, the `aggregation_factor` and the `habitat_threshold` parameters, described below, will be used to down sample the habitat raster to a tractable resolution for the optimization engine, and automatically derive the restorable habitat raster.

`habitat_threshold`
: `numeric` number between 0 and 1, which corresponds to the minimum proportion of habitat that must be present within an aggregated pixel to consider it as an habitat pixel.

`aggregation_factor`
: `integer` Integer greater than 1, which corresponds to the aggregation factor for down sampling the data. For example, if `aggregation_factor = 2`, aggregated pixel will contain 4 original pixel. See `terra::aggregate()` for more details.

Details

This function creates the base restoration optimization problem object, that can be further extended with constraints and optimization objectives. One input rasters is necessary to instantiate a restopt problem: the `existing_habitat` raster (potentially with high resolution). This raster must contains data about where are habitat areas (raster value `1`), non-habitat areas (raster value `0`), and areas that must not be considered during the solving procedure (`NA` or `NO_DATA`). The `aggregation_factor` parameter is used to down sample the `existing_habitat` to a resolution

that will be tractable for the optimization engine, and the habitat_threshold parameter indicates the minimum proportion of habitat required in aggregated habitat pixels to consider them as habitat. Note that An aggregated pixel will contain at most aggregation_factor^2 pixels from the input habitat raster (cell_area raster in this function outputs). If an aggregated pixel is close to the spatial boundaries of the problem (i.e. NA cells), it can contain less than aggregation_factor^2 fine grained pixel. You can get the results of this preprocessing phase using the following methods: get_original_habitat() (original habitat), get_existing_habitat() (aggregated habitat), get_cell_area() (number of pixels in each aggregated cells), and get_restorable_area() (amount of restorable area – in number of original raster pixels).

Value

A new restoration problem (RestoptProblem) object.

None.

Examples

```
# load data
habitat_data <- rast(
  system.file("extdata", "habitat_hi_res.tif", package = "restoptr")
)

# create problem
p <- restopt_problem(
       existing_habitat = habitat_data,
       aggregation_factor = 4,
       habitat_threshold = 0.7
)

# Plot down sampled data
plot(c(p$data$existing_habitat, p$data$restorable_habitat))

# print problem
print(p)
```

---

restopt_solution *Restopt solution*

Description

An object representing a restopt problem solution. It is basically a SpatRaster with a few metadata attributes added.

Usage

```
restopt_solution(restopt_problem, solution_raster, metadata, id_solution = 1)
```

Arguments

`restopt_problem`	`restopt_problem()` Reference to the problem corresponding to this solution.
`solution_raster`	`terra::rast()` Solution raster.
`metadata`	`list` List containing metadata attributes of the solution.
`id_solution`	`integer` Identifier of the solution.

Value

A new restoration problem solution (`restopt_solution()`).

`set_max_iic_objective` *Set an objective to maximize the integral index of connectivity*

Description

Specify that a restoration problem (`restopt_problem()`) should the integral index of connectivity (IIC).

Usage

```
set_max_iic_objective(problem, distance_threshold = -1, unit = "m")
```

Arguments

`problem`	`restopt_problem()` Restoration problem object.
`distance_threshold`	`numeric` greater than 0. Minimum distance (in `unit`) between two patches to consider them connected in the computation of the IIC. The default value -1 causes the function to use 1 aggregated cell as the distance threshold.
`unit`	`unit` object or a `character` that can be coerced to a distance unit (see `unit` package), or "cells" for cell width of aggregated habitat raster. Units of the `distance_threshold` parameter. If the input habitat raster does not use a projected coordinate system, only "cells" is available. Meters by default, expected if `distance_threshold` is set to its default value (-1), which causes the function to use 1 cell by default.

Details

The integral index of connectivity (IIC) is a graph-based inter-patch connectivity index based on a binary connection model (Pascual-Hortal & Saura, 2006). Its maximization in the context of restoration favours restoring the structural connectivity between large patches. IIC is unitless and comprised between 0 (no connectivity) and 1 (all the landscape is habitat, thus fully connected). The `distance_threshold` parameter indicates to the solver how to construct the habitat graph, i.e. what is the minimum distance between two patches to consider them as connected. Note that, as the computation occurs on aggregated cells, if `distance_threshold` is used with a different unit than "cells", it will be rounded to the closest corresponding number of cells.

Value

An updated restoration problem (restopt_problem() object.

References

Pascual-Hortal, L., & Saura, S. (2006). Comparison and development of new graph-based landscape connectivity indices: Towards the priorization of habitat patches and corridors for conservation. Landscape Ecology, 21(7), 959-967. https://doi.org/10.1007/s10980-006-0013-z

See Also

Other objectives: set_max_mesh_objective(), set_max_nb_pus_objective(), set_max_restore_objective(), set_min_nb_pus_objective(), set_min_restore_objective(), set_no_objective()

Examples

```
# load data
habitat_data <- rast(
  system.file("extdata", "habitat_hi_res.tif", package = "restoptr")
)

locked_out_data <- rast(
 system.file("extdata", "locked_out.tif", package = "restoptr")
)

# create problem with locked out constraints
p <- restopt_problem(
    existing_habitat = habitat_data,
    aggregation_factor = 16,
    habitat_threshold = 0.7
  ) %>%
  set_max_iic_objective() %>%
  add_restorable_constraint(
    min_restore = 5,
    max_restore = 5,
  ) %>%
  add_locked_out_constraint(data = locked_out_data) %>%
  add_settings(time_limit = 1)

# print problem
print(p)

# solve problem
s <- solve(p)

# plot solution
plot(s)
```

set_max_mesh_objective *Set an objective to maximize effective mesh size*

Description

Specify that a restoration problem (restopt_problem()) should maximize effective mesh size.

Usage

```
set_max_mesh_objective(problem)
```

Arguments

problem	restopt_problem() Restoration problem object.

Details

The effective mesh size (MESH) is a measure of landscape fragmentation based on the probability that two randomly chosen points are located in the same patch (Jaeger, 2000). Maximizing it in the context of restoration favours fewer and larger patches.

Value

An updated restoration problem (restopt_problem()) object.

References

Jaeger, J. A. G. (2000). Landscape division, splitting index, and effective mesh size: New measures of landscape fragmentation. Landscape Ecology, 15(2), 115-130. https://doi.org/10.1023/A:1008129329289

See Also

Other objectives: set_max_iic_objective(), set_max_nb_pus_objective(), set_max_restore_objective(), set_min_nb_pus_objective(), set_min_restore_objective(), set_no_objective()

Examples

```
# load data
habitat_data <- rast(
  system.file("extdata", "habitat_hi_res.tif", package = "restoptr")
)

locked_out_data <- rast(
 system.file("extdata", "locked_out.tif", package = "restoptr")
)

# plot data
```

```
plot(rast(list(habitat_data, locked_out_data)), nc = 2)

# create problem with locked out constraints
p <- restopt_problem(
    existing_habitat = habitat_data,
    aggregation_factor = 16,
    habitat_threshold = 0.7
  ) %>%
  set_max_mesh_objective() %>%
  add_restorable_constraint(
    min_restore = 5,
    max_restore = 5,
  ) %>%
  add_locked_out_constraint(data = locked_out_data) %>%
  add_settings(time_limit = 1)

# print problem
print(p)

# solve problem
s <- solve(p)

# plot solution
plot(s)
```

`set_max_nb_pus_objective` *Set an objective to maximize the number of planning units*

Description

Specify that a restoration problem (`restopt_problem()`) should maximize the number of planning units.

Usage

```
set_max_nb_pus_objective(problem)
```

Arguments

`problem`	`restopt_problem()`	Restoration problem object.

Details

Planning units correspond to aggregated cells from the original dataset. Maximizing the number of planning units increased the spatial extent of the restoration area. This can be useful when the budget is limited and the aim is to restore the larget possible extent.

Value

An updated restoration problem (restopt_problem()) object.

See Also

Other objectives: set_max_iic_objective(), set_max_mesh_objective(), set_max_restore_objective(), set_min_nb_pus_objective(), set_min_restore_objective(), set_no_objective()

Examples

```
# load data
habitat_data <- rast(
  system.file("extdata", "habitat_hi_res.tif", package = "restoptr")
)

locked_out_data <- rast(
 system.file("extdata", "locked_out.tif", package = "restoptr")
)

# plot data
plot(rast(list(habitat_data, locked_out_data)), nc = 2)

# create problem with locked out constraints
p <- restopt_problem(
    existing_habitat = habitat_data,
    aggregation_factor = 16,
    habitat_threshold = 0.7
  ) %>%
  set_max_nb_pus_objective() %>%
  add_restorable_constraint(
    min_restore = 5,
    max_restore = 5,
  ) %>%
  add_locked_out_constraint(data = locked_out_data) %>%
  add_settings(time_limit = 1)

# print problem
print(p)

# solve problem
s <- solve(p)

# plot solution
plot(s)
```

set_max_restore_objective *Set an objective to maximize the amount restoration area.*

Description

Specify that a restoration problem (restopt_problem()) should maximize the restoration area needed to reach the habitat proportion threshold specified in the problem description.

Usage

```
set_max_restore_objective(problem)
```

Arguments

problem	restopt_problem() Restoration problem object.

Details

The restoration area corresponds to the minimum amount of area that must be restored in the selected planning units to reach the minimum habitat proportion threshold specified in the problem description,

Value

An updated restoration problem (restopt_problem()) object.

See Also

Other objectives: set_max_iic_objective(), set_max_mesh_objective(), set_max_nb_pus_objective(), set_min_nb_pus_objective(), set_min_restore_objective(), set_no_objective()

Examples

```
# load data
habitat_data <- rast(
  system.file("extdata", "habitat_hi_res.tif", package = "restoptr")
)

locked_out_data <- rast(
 system.file("extdata", "locked_out.tif", package = "restoptr")
)

# plot data
plot(rast(list(habitat_data, locked_out_data)), nc = 2)

# create problem with locked out constraints
p <- restopt_problem(
```

```
    existing_habitat = habitat_data,
    aggregation_factor = 16,
    habitat_threshold = 0.7
  ) %>%
  set_max_restore_objective() %>%
  add_restorable_constraint(
    min_restore = 5,
    max_restore = 5,
  ) %>%
  add_locked_out_constraint(data = locked_out_data) %>%
  add_settings(time_limit = 1)

# print problem
print(p)

# solve problem
s <- solve(p)

# plot solution
plot(s)
```

`set_min_nb_pus_objective` *Set an objective to minimize the number of planning units*

Description

Specify that a restoration problem (`restopt_problem()`) should minimize the number of planning units.

Usage

```
set_min_nb_pus_objective(problem)
```

Arguments

`problem`	`restopt_problem()` Restoration problem object.

Details

Planning units correspond to aggregated cells from the original dataset. Minimizing the number of planning units reduces the spatial extent of the restoration area.

Value

An updated restoration problem (`restopt_problem()`) object.

See Also

Other objectives: set_max_iic_objective(), set_max_mesh_objective(), set_max_nb_pus_objective(), set_max_restore_objective(), set_min_restore_objective(), set_no_objective()

Examples

```
# load data
habitat_data <- rast(
  system.file("extdata", "habitat_hi_res.tif", package = "restoptr")
)

locked_out_data <- rast(
 system.file("extdata", "locked_out.tif", package = "restoptr")
)

# plot data
plot(rast(list(habitat_data, locked_out_data)), nc = 2)

# create problem with locked out constraints
p <- restopt_problem(
    existing_habitat = habitat_data,
    aggregation_factor = 16,
    habitat_threshold = 0.7
  ) %>%
  set_min_nb_pus_objective() %>%
  add_restorable_constraint(
    min_restore = 5,
    max_restore = 5,
  ) %>%
  add_locked_out_constraint(data = locked_out_data) %>%
  add_settings(time_limit = 1)

# print problem
print(p)

# solve problem
s <- solve(p)

# plot solution
plot(s)
```

`set_min_restore_objective` *Set an objective to minimize the amount restoration area.*

Description

Specify that a restoration problem (restopt_problem()) should minimize the restoration area needed to reach the habitat proportion threshold specified in the problem description.

Usage

```
set_min_restore_objective(problem)
```

Arguments

problem	restopt_problem() Restoration problem object.

Details

The restoration area corresponds to the minimum amount of area that must be restored in the selected planning units to reach the minimum habitat proportion threshold specified in the problem description,

Value

An updated restoration problem (restopt_problem()) object.

See Also

Other objectives: set_max_iic_objective(), set_max_mesh_objective(), set_max_nb_pus_objective(), set_max_restore_objective(), set_min_nb_pus_objective(), set_no_objective()

Examples

```
# load data
habitat_data <- rast(
  system.file("extdata", "habitat_hi_res.tif", package = "restoptr")
)

locked_out_data <- rast(
 system.file("extdata", "locked_out.tif", package = "restoptr")
)

# plot data
plot(rast(list(habitat_data, locked_out_data)), nc = 2)

# create problem with locked out constraints
p <- restopt_problem(
    existing_habitat = habitat_data,
    aggregation_factor = 16,
    habitat_threshold = 0.7
  ) %>%
  set_min_restore_objective() %>%
  add_restorable_constraint(
    min_restore = 5,
```

```
    max_restore = 5,
  ) %>%
  add_locked_out_constraint(data = locked_out_data) %>%
  add_settings(time_limit = 1)

# print problem
print(p)

# solve problem
s <- solve(p)

# plot solution
plot(s)
```

`set_no_objective` *Configure the solver to only satisfy the constraints, without optimization objective*

Description

Specify that a restoration problem (`restopt_problem()`) should satisfy the constraints without optimization objective.

Usage

```
set_no_objective(problem)
```

Arguments

`problem`	`restopt_problem()` Restoration problem object.

Details

Using set_no_objective() in a restopt problem, the solver will return the first solution found satisfying the constraint, without any optimization objective. This "no objective" setting is set by default when creating a restopt problem.

Value

An updated restoration problem (`restopt_problem()`) object.

See Also

Other objectives: `set_max_iic_objective()`, `set_max_mesh_objective()`, `set_max_nb_pus_objective()`, `set_max_restore_objective()`, `set_min_nb_pus_objective()`, `set_min_restore_objective()`

Examples

```
# load data
habitat_data <- rast(
  system.file("extdata", "habitat_hi_res.tif", package = "restoptr")
)

# create problem
p <- restopt_problem(
    existing_habitat = habitat_data,
    aggregation_factor = 16,
    habitat_threshold = 0.7
  ) %>% set_no_objective()

# print problem
print(p)

# Solve problem
s <- solve(p)
# plot solution
plot(s)
```

`solve.RestoptProblem` *Solve a restoration optimization problem*

Description

Solve a restoration optimization problem to generate a solution.

Usage

```
## S3 method for class 'RestoptProblem'
solve(a, b, ...)
```

Arguments

a	`restopt_problem()` Restoration problem object.
b	Argument not used.
...	Additional arguments: `verbose`: if TRUE, output solver logs. (FALSE by default)

Details

This function relies on the Choco-solver (https://choco-solver.org/) to solve a restoration optimization problem. If the solver finds a solution, it outputs a raster with 5 possible values: NA : NA (or NO_DATA) areas from the input habitat raster. 0 : non-habitat areas that were locked out. 1

: non-habitat areas that were available for selection. 2 : habitat areas. 3 : selected planning units for restoration. If the solve function return a no-solution error, it is either because the solver could not find a solution within the time limit that was set (see add_settings), or because the solver has detected that this is not possible to satisfy the constraints (the constraints are contradictory). In the first case, you can try to increase the time limit. In the second case, you should modify your targets.

Value

A `restopt_solution()` object.

Examples

```
# load data
habitat_data <- rast(
  system.file("extdata", "habitat_hi_res.tif", package = "restoptr")
)

available <- vect(
  system.file("extdata", "accessible_areas.gpkg", package = "restoptr")
)

# create problem with locked out constraints
p <- restopt_problem(
    existing_habitat = habitat_data,
    aggregation_factor = 16,
    habitat_threshold = 0.7
  ) %>%
  set_max_mesh_objective() %>%
  add_restorable_constraint(
    min_restore = 5,
    max_restore = 5,
  ) %>%
  add_available_areas_constraint(available) %>%
  add_settings(time_limit = 1)

# print problem
print(p)

# solve problem
s <- solve(p)

# plot solution
plot(s)
```

Index