

# Package ‘raptr’

July 25, 2022

**Type** Package

**Title** Representative and Adequate Prioritization Toolkit in R

**Version** 0.2.1

**Description** Biodiversity is in crisis. The overarching aim of conservation is to preserve biodiversity patterns and processes. To this end, protected areas are established to buffer species and preserve biodiversity processes. But resources are limited and so protected areas must be cost-effective. This package contains tools to generate plans for protected areas (prioritizations), using spatially explicit targets for biodiversity patterns and processes. To obtain solutions in a feasible amount of time, this package uses the commercial 'Gurobi' software (obtained from <https://www.gurobi.com/>). For more information on using this package, see Hanson et al. (2018) <[doi:10.1111/2041-210X.12862](https://doi.org/10.1111/2041-210X.12862)>.

**Imports** utils, methods, stats, graphics, grDevices, parallel, Matrix (>= 1.4.1), assertthat (>= 0.2.1), boot (>= 1.3.28), PBSmapping (>= 2.73.0), scales (>= 1.2.0), shape (>= 1.4.6), adehabitatHR (>= 0.4.19), RColorBrewer (>= 1.1.3), ggplot2 (>= 3.3.5), plyr (>= 1.8.7), doParallel (>= 1.0.17), rgeos (>= 0.5.9), rgdal (>= 1.5.30), hypervolume (>= 2.0.7), ks (>= 1.13.5), mvtnorm (>= 1.1.3), withr (>= 2.5.0),

**Depends** R (>= 3.5.0), sp (>= 1.4.6), raster (>= 3.5.15)

**LinkingTo** Rcpp, RcppEigen, BH

**LazyData** true

**License** GPL-3

**Encoding** UTF-8

**Language** en-US

**URL** <https://jeffrey-hanson.com/raptr/>,  
<https://github.com/jeffreyhanson/raptr>

**BugReports** <https://github.com/jeffreyhanson/raptr/issues>

**VignetteBuilder** knitr

**Suggests** knitr, roxygen2, rmarkdown, testthat, RgoogleMaps (>= 1.4.5.3), dplyr (>= 1.0.8), vegan (>= 2.6.2), gurobi (>= 8.0.0), gridExtra (>= 2.3), rgl (>= 0.108.3),

**SystemRequirements** C++11

**Collate** 'dependencies.R' 'RcppExports.R' 'raptr-internal.R' 'generics.R' 'DemandPoints.R' 'misc.R' 'PlanningUnitPoints.R' 'AttributeSpace.R' 'AttributeSpaces.R' 'GurobiOpts.R' 'ManualOpts.R' 'calcSpeciesAverageInPus.R' 'calcBoundaryData.R' 'RapData.R' 'RapReliableOpts.R' 'RapResults.R' 'RapUnreliableOpts.R' 'RapUnsolved.R' 'RapSolved.R' 'SpatialPolygons2PolySet.R' 'data.R' 'rap.R' 'raptr.R' 'rrap.proportion.held.R' 'sim.pus.R' 'sim.space.R' 'sim.species.R' 'urap.proportion.held.R' 'zzz.R'

**RoxygenNote** 7.2.1

**NeedsCompilation** yes

**Author** Jeffrey O Hanson [aut, cre],  
Jonathan R Rhodes [aut],  
Hugh P Possingham [aut],  
Richard A Fuller [aut]

**Maintainer** Jeffrey O Hanson <jeffrey.hanson@uqconnect.edu.au>

**Repository** CRAN

**Date/Publication** 2022-07-25 14:10:02 UTC

## R topics documented:

amount.held . . . . .	4
amount.target . . . . .	5
as.list . . . . .	6
AttributeSpace . . . . .	7
AttributeSpace-class . . . . .	7
AttributeSpaces . . . . .	8
AttributeSpaces-class . . . . .	9
blank.raster . . . . .	9
calcBoundaryData . . . . .	10
calcSpeciesAverageInPus . . . . .	11
casestudy_data . . . . .	12
DemandPoints . . . . .	13
DemandPoints-class . . . . .	14
dp.subset . . . . .	14
GurobiOpts . . . . .	15
GurobiOpts-class . . . . .	16
is.GurobiInstalled . . . . .	17
logging.file . . . . .	18
make.DemandPoints . . . . .	19
make.RapData . . . . .	20

ManualOpts . . . . .	22
ManualOpts-class . . . . .	23
maximum.targets . . . . .	23
names . . . . .	24
PlanningUnitPoints . . . . .	25
PlanningUnitPoints-class . . . . .	25
plot . . . . .	26
PolySet-class . . . . .	28
print . . . . .	28
prob.subset . . . . .	29
pu.subset . . . . .	30
randomPoints . . . . .	31
rap . . . . .	32
RapData . . . . .	33
RapData-class . . . . .	35
RapOpts-class . . . . .	35
RapReliableOpts . . . . .	36
RapReliableOpts-class . . . . .	36
RapResults . . . . .	37
RapResults-class . . . . .	38
RapSolved . . . . .	39
RapSolved-class . . . . .	40
raptr . . . . .	40
RapUnreliableOpts . . . . .	41
RapUnreliableOpts-class . . . . .	42
RapUnsolved . . . . .	42
RapUnsolved-class . . . . .	43
rrap.proportion.held . . . . .	43
score . . . . .	44
selections . . . . .	45
show . . . . .	46
sim.pus . . . . .	48
sim.space . . . . .	49
sim.species . . . . .	50
simulated_data . . . . .	51
solve . . . . .	52
SolverOpts-class . . . . .	54
space.held . . . . .	54
space.plot . . . . .	55
space.target . . . . .	56
SpatialPolygons2PolySet . . . . .	58
spp.plot . . . . .	59
spp.subset . . . . .	61
summary . . . . .	62
update . . . . .	63
urap.proportion.held . . . . .	66

---

amount.held	<i>Extract amount held for a solution</i>
-------------	---

---

### Description

This function returns the amount held for each species in a solution.

### Usage

```
amount.held(x, y, species)

## S3 method for class 'RapSolved'
amount.held(x, y = 0, species = NULL)
```

### Arguments

x	<a href="#">RapResults()</a> or <a href="#">RapSolved()</a> object.
y	Available inputs include: NULL to return all values, integer number specifying the solution for which the value should be returned, and 0 to return the value for the best solution.
species	NULL for all species or integer indicating species.

### Value

[base::matrix\(\)](#) or numeric vector depending on arguments.

### See Also

[RapResults\(\)](#), [RapSolved\(\)](#).

### Examples

```
# load data
data(sim_rs)

# amount held (%) in best solution for each species
amount.held(sim_rs, 0)

# amount held (%) in best solution for species 1
amount.held(sim_rs, 0, 1)

# amount held (%) in second solution for each species
amount.held(sim_rs, 2)

# amount held (%) in each solution for each species
amount.held(sim_rs, NULL)
```

---

amount.target	<i>Amount targets</i>
---------------	-----------------------

---

## Description

This function sets or returns the target amounts for each species.

## Usage

```
amount.target(x, species)

amount.target(x, species) <- value

## S3 method for class 'RapData'
amount.target(x, species = NULL)

## S3 replacement method for class 'RapData'
amount.target(x, species = NULL) <- value

## S3 method for class 'RapUnsolved'
amount.target(x, species = NULL)

## S3 replacement method for class 'RapUnsolved'
amount.target(x, species = NULL) <- value
```

## Arguments

x	<a href="#">RapData()</a> , <a href="#">RapUnsolved()</a> , or <a href="#">RapSolved()</a> object.
species	NULL for all species or integer indicating species.
value	numeric new target.

## Value

codenumeric vector.

## See Also

[RapData\(\)](#), [RapResults\(\)](#), [RapSolved\(\)](#).

## Examples

```
# load data
data(sim_rs)

# extract amount targets for all species
amount.target(sim_rs)
```

```
# set amount targets for all species
amount.target(sim_rs) <- 0.1

# extract amount targets for first species
amount.target(sim_rs, 1)

# set amount targets for for first species
amount.target(sim_rs, 1) <- 0.5
```

---

as.list

*Convert object to list*

---

### Description

Convert `GurobiOpts()` object to list.

### Usage

```
## S3 method for class 'GurobiOpts'
as.list(x, ...)
```

### Arguments

x	<code>GurobiOpts()</code> object.
...	not used.

### Value

list

### Note

This function will not include the `NumberSolutions` slot, the `MultipleSolutionsMethod` slot, or the `TimeLimit` slot if it is not finite.

### See Also

`GurobiOpts`.

### Examples

```
# make GuboriOpts object
x <- GurobiOpts()

# convert to list
as.list(x)
```

---

AttributeSpace	Create new AttributeSpace object
----------------	----------------------------------

---

### Description

This function creates a new AttributeSpace object.

### Usage

```
AttributeSpace(planning.unit.points, demand.points, species)
```

### Arguments

`planning.unit.points` [PlanningUnitPoints\(\)](#) for planning unit in the space.  
`demand.points` [DemandPoints\(\)](#) object for the space.  
`species` integer species id to indicate which species the space is associated with.

### See Also

[DemandPoints](#), [PlanningUnitPoints](#).

### Examples

```
space <- AttributeSpace(  
  PlanningUnitPoints(  
    matrix(rnorm(100), ncol = 2),  
    seq_len(50)),  
  DemandPoints(  
    matrix(rnorm(100), ncol = 2),  
    runif(50)),  
  species = 1L)
```

---

AttributeSpace-class	AttributeSpace: An S4 class to represent an attribute space.
----------------------	--

---

### Description

This class is used to store planning unit points and demand points for a single species in an attribute space.

### Slots

`planning.unit.points` [PlanningUnitPoints\(\)](#) for planning unit in the space.  
`demand.points` [DemandPoints\(\)](#) object for the space.  
`species` integer species id to indicate which species the space is associated with.

**See Also**

[DemandPoints](#), [PlanningUnitPoints](#).

---

AttributeSpaces      *Create new AttributeSpaces object*

---

**Description**

This function creates a new AttributeSpaces object.

**Usage**

```
AttributeSpaces(spaces, name)
```

**Arguments**

spaces	list of <a href="#">AttributeSpace()</a> objects for different species.
name	character name to identify the attribute space.

**See Also**

[AttributeSpace](#).

**Examples**

```
space1 <- AttributeSpace(  
  PlanningUnitPoints(  
    matrix(rnorm(100), ncol = 2),  
    seq_len(50)),  
  DemandPoints(  
    matrix(rnorm(100), ncol = 2),  
    runif(50)),  
  species = 1L)  
  
space2 <- AttributeSpace(  
  PlanningUnitPoints(  
    matrix(rnorm(100), ncol = 2),  
    seq_len(50)),  
  DemandPoints(  
    matrix(rnorm(100), ncol = 2),  
    runif(50)),  
  species = 2L)  
  
spaces <- AttributeSpaces(list(space1, space2), "spaces")
```

---

AttributeSpaces-class *AttributeSpaces*: An S4 class to represent a collection of attribute spaces for different species.

---

### Description

This class is used to store a collection of attribute spaces for different species.

### Slots

spaces list of [AttributeSpace\(\)](#) objects for different species.  
name character name to identify the attribute space.

### See Also

[AttributeSpace](#).

---

blank.raster                      *Blank raster*

---

### Description

This functions creates a blank raster based on the spatial extent of a Spatial object.

### Usage

```
blank.raster(x, res)
```

### Arguments

x                      [sp::Spatial](#) object.  
res                     numeric vector specifying resolution of the output raster in the x and y dimensions. If vector is of length one, then the pixels are assumed to be square.

### Examples

```
# make SpatialPolygons  
polys <- sim.pus(225L)  
  
# make RasterLayer from SpatialPolygons  
blank.raster(polys, 1)
```

---

calcBoundaryData      *Calculate boundary data for planning units*

---

### Description

This function calculates boundary length data for `PBSmapping::PolySet()`, `sp::SpatialPolygons()`, and `sp::SpatialPolygonsDataFrame()` objects. Be aware that this function is designed with performance in mind, and as a consequence, if this function is used improperly then it may crash R. Furthermore, multipart polygons with touching edges will likely result in inaccuracies. If argument set to `sp::SpatialPolygons()` or `sp::SpatialPolygonsDataFrame()`, this will be converted to `PolySet` before processing.

### Usage

```
calcBoundaryData(x, tol, length.factor, edge.factor)

## S3 method for class 'PolySet'
calcBoundaryData(x, tol = 0.001, length.factor = 1, edge.factor = 1)

## S3 method for class 'SpatialPolygons'
calcBoundaryData(x, tol = 0.001, length.factor = 1, edge.factor = 1)
```

### Arguments

<code>x</code>	<code>PBSmapping::PolySet()</code> , <code>sp::SpatialPolygons()</code> or <code>sp::SpatialPolygonsDataFrame()</code> object.
<code>tol</code>	numeric to specify precision of calculations. In other words, how far apart vertices have to be to be considered different?
<code>length.factor</code>	numeric to scale boundary lengths.
<code>edge.factor</code>	numeric to scale boundary lengths for edges that do not have any neighbors, such as those that occur along the margins.

### Value

data.frame with 'id1' (integer), 'id2' (integer), and 'amount' (numeric) columns.

### See Also

This function is based on the algorithm in QMARXAN <https://github.com/tsw-afropos/qmarxan> for calculating boundary length.

### Examples

```
# simulate planning units
sim_pus <- sim.pus(225L)

# calculate boundary data
```

```
bound.dat <- calcBoundaryData(sim_pus)

# print summary of boundary data
summary(bound.dat)
```

---

calcSpeciesAverageInPus

*Calculate average value for species data in planning units*

---

### Description

This function calculates the average of species values in each planning unit. By default all polygons will be treated as having separate ids.

### Usage

```
calcSpeciesAverageInPus(x, ...)

## S3 method for class 'SpatialPolygons'
calcSpeciesAverageInPus(x, y, ids = seq_len(nlayers(y)), ncores = 1, ...)

## S3 method for class 'SpatialPolygonsDataFrame'
calcSpeciesAverageInPus(
  x,
  y,
  ids = seq_len(nlayers(y)),
  ncores = 1,
  field = NULL,
  ...
)
```

### Arguments

x	<a href="#">sp::SpatialPolygons()</a> or <a href="#">sp::SpatialPolygonsDataFrame()</a> object.
...	not used.
y	<a href="#">raster::raster()</a> , <a href="#">raster::stack()</a> , or <a href="#">raster::brick()</a> object.
ids	integer vector of ids. Defaults to indices of layers in argument to y.
ncores	integer number of cores to use for processing. Defaults to 1.
field	integer index or character name of column with planning unit ids. Valid only when x is a <a href="#">sp::SpatialPolygonsDataFrame()</a> object. Default behavior is to treat each polygon as a different planning unit.

### Value

[base::data.frame\(\)](#) with sum of raster values in each polygon.

## Examples

```
# simulate data
sim_pus <- sim.pus(225L)
sim_spp <- lapply(c("uniform", "normal", "bimodal"), sim.species, n = 1,
                 res = 1, x = sim_pus)

# calculate average for 1 species
puvspr1.dat <- calcSpeciesAverageInPus(sim_pus, sim_spp[[1]])

# calculate average for multiple species
puvspr2.dat <- calcSpeciesAverageInPus(sim_pus, stack(sim_spp))
```

---

casestudy\_data

*Case-study dataset for a conservation planning exercise*

---

## Description

This dataset contains data to generate example prioritizations for the pale-headed Rosella (*Platycercus adscitus*) in Queensland, Australia.

## Format

**cs\_pus** `sp::SpatialPolygonsDataFrame()` object.

**cs\_spp** `raster::raster()` object.

**cs\_space** `raster::stack()` object.

## Details

The objects in the dataset are listed below.

**cs\_pus** `sp::SpatialPolygonsDataFrame()` planning units. The units were generated as  $30\text{km}^2$  squares across the species' range, and then clipped to the Queensland, Australia (using data obtained from the Australia Bureau of Statistics; <https://www.abs.gov.au/ausstats/abs@.nsf/mf/1259.0.30.001?OpenDocument>). They were then overlaid with Australia's protected area network (obtained from the World Database on Protected Areas (WDPA) at <https://www.protectedplanet.net/en>). This attribute table has 3 fields. The area field denotes the amount of land encompassed by each unit, the cost field is set to 1 for all units, and the status field indicates if 50% or more of the units' extent is covered by protected areas.

**cs\_spp** `raster::raster()` probability distribution map for the *P. adscitus* clipped to Queensland, Australia. This map was derived from records obtained from The Atlas of Living Australia (<https://ala.org.au/>).

**cs\_space** `raster::stack()` describing broad-scale climate variation across Queensland (obtained from <https://worldclim.org/>, and resampled to  $10\text{km}^2$  resolution).

**Examples**

```
# load data
data(cs_pus, cs_spp, cs_space)

# plot data
## Not run:
plot(cs_pus)
plot(cs_spp)
plot(cs_space)

## End(Not run)
```

---

DemandPoints	<i>Create new DemandPoints object</i>
--------------	---------------------------------------

---

**Description**

This function creates a new DemandPoints object

**Usage**

```
DemandPoints(coords, weights)
```

**Arguments**

coords	<a href="#">base::matrix()</a> of coordinates for each demand point.
weights	numeric weights for each demand point.

**See Also**

[DemandPoints.](#)

**Examples**

```
# make demand points
dps <- DemandPoints(
  matrix(rnorm(100), ncol=2),
  runif(50))

# print object
print(dps)
```

---

DemandPoints-class      *DemandPoints: An S4 class to represent demand points*

---

### Description

This class is used to store demand point information.

### Slots

coords `base::matrix()` of coordinates for each demand point.

weights numeric weights for each demand point.

### See Also

[DemandPoints\(\)](#).

---

dp.subset      *Subset demand points*

---

### Description

Subset demand points from a [RapData\(\)](#), [RapUnsolved\(\)](#), or [RapSolved\(\)](#) object.

### Usage

```
dp.subset(x, space, species, points)
```

```
## S3 method for class 'RapData'
dp.subset(x, space, species, points)
```

```
## S3 method for class 'RapUnsolvedOrSolved'
dp.subset(x, space, species, points)
```

### Arguments

x	<a href="#">RapData()</a> , <a href="#">RapUnsolved()</a> , or <a href="#">RapSolved()</a> object.
space	integer vector to specify the index of the space to subset demand points from.
species	integer vector to specify the index of the species to subset demand points from.
points	integer vector to specify the index of demand points to subset.

### Value

[RapData\(\)](#) or [RapUnsolved\(\)](#) object depending on input object.

**See Also**

[RapData\(\)](#), [RapUnsolved\(\)](#), [RapSolved\(\)](#).

**Examples**

```
# load data
data(sim_ru)

# generate new object with first 10 planning units
sim_ru2 <- dp.subset(sim_ru, 1, 1, seq_len(10))
```

---

GurobiOpts

*Create GurobiOpts object*


---

**Description**

This function creates a new GurobiOpts object.

**Usage**

```
GurobiOpts(
  Threads = 1L,
  MIPGap = 0.1,
  Method = 0L,
  Presolve = 2L,
  TimeLimit = NA_integer_,
  NumberSolutions = 1L,
  MultipleSolutionsMethod = c("benders.cuts", "solution.pool.0", "solution.pool.1",
    "solution.pool.2")[1]
)
```

**Arguments**

Threads	integer number of cores to use for processing. Defaults to 1L.
MIPGap	numeric MIP gap specifying minimum solution quality. Defaults to 0.1.
Method	integer Algorithm to use for solving model. Defaults to 0L.
Presolve	integer code for level of computation in presolve (lp_solve parameter). Defaults to 2.
TimeLimit	integer number of seconds to allow for solving. Defaults to NA_integer_, and so a time limit is not imposed.
NumberSolutions	integer number of solutions to generate. Defaults to 1L.

**MultipleSolutionsMethod**

integer name of method to obtain multiple solutions (used when `NumberSolutions` is greater than one). Available options are "benders.cuts", "solution.pool.0", "solution.pool.1", and "solution.pool.2". The "benders.cuts" method produces a set of distinct solutions that are all within the optimality gap. The "solution.pool.0" method returns all solutions identified whilst trying to find a solution that is within the specified optimality gap. The "solution.pool.1" method finds one solution within the optimality gap and a number of additional solutions that are of any level of quality (such that the total number of solutions is equal to `number_solutions`). The "solution.pool.2" finds a specified number of solutions that are nearest to optimality. The search pool methods correspond to the parameters used by the Gurobi software suite (see <https://www.gurobi.com/documentation/8.0/refman/poolsearchmode.html#parameter:PoolSearchMode>). Defaults to "benders.cuts".

**Value**

GurobiOpts object

**See Also**

[GurobiOpts](#).

**Examples**

```
# create GurobiOpts object using default parameters
GurobiOpts(Threads = 1L, MIPGap = 0.1, Method = 0L, Presolve=2L,
           TimeLimit = NA_integer_, NumberSolutions = 1L)
```

---

GurobiOpts-class

*GurobiOpts: An S4 class to represent Gurobi parameters*

---

**Description**

This class is used to store Gurobi input parameters.

**Slots**

`Threads` integer number of cores to use for processing. Defaults to 1L.

`MIPGap` numeric MIP gap specifying minimum solution quality. Defaults to 0.1.

`Method` integer Algorithm to use for solving model. Defaults to 0L.

`Presolve` integer code for level of computation in presolve. Defaults to 2.

`TimeLimit` integer number of seconds to allow for solving. Defaults to `NA_integer_`, and so a time limit is not imposed.

`NumberSolutions` integer number of solutions to generate. Defaults to 1L.

MultipleSolutionsMethod integer name of method to obtain multiple solutions (used when NumberSolutions is greater than one). Available options are "benders.cuts", "solution.pool.0", "solution.pool.1", and "solution.pool.2". The "benders.cuts" method produces a set of distinct solutions that are all within the optimality gap. The "solution.pool.0" method returns all solutions identified whilst trying to find a solution that is within the specified optimality gap. The "solution.pool.1" method finds one solution within the optimality gap and a number of additional solutions that are of any level of quality (such that the total number of solutions is equal to number\_solutions). The "solution.pool.2" finds a specified number of solutions that are nearest to optimality. The search pool methods correspond to the parameters used by the Gurobi software suite (see <https://www.gurobi.com/documentation/8.0/refman/poolsearchmode.html#parameter:PoolSearchMode>). Defaults to "benders.cuts".

### See Also

[GurobiOpts\(\)](#).

---

is.GurobiInstalled      *Test if Gurobi is installed*

---

### Description

This function determines if the Gurobi R package is installed on the computer and that it can be used [base::options\(\)](#).

### Usage

```
is.GurobiInstalled(verbose = TRUE)
```

### Arguments

verbose            logical should messages be printed?

### Value

logical Is it installed and ready to use?

### See Also

[base::options\(\)](#).

### Examples

```
## Not run:
# check if Gurobi is installed
is.GurobiInstalled()

# print cached status of installation
options()$GurobiInstalled
```

```
## End(Not run)
```

---

logging.file	<i>Log file</i>
--------------	-----------------

---

### Description

This function returns the Gurobi log file (\*.log) associated with solving an optimization problem.

### Usage

```
logging.file(x, y)

## S3 method for class 'RapResults'
logging.file(x, y = 0)

## S3 method for class 'RapSolved'
logging.file(x, y = 0)
```

### Arguments

x	<a href="#">RapResults()</a> or <a href="#">RapSolved()</a> object.
y	Available inputs include: NULL to return all values, integer number specifying the solution for which the log file should be returned, and 0 to return log file for the best solution.

### Note

The term logging file was used due to collisions with the log function.

### See Also

[RapResults\(\)](#), [RapSolved\(\)](#).

### Examples

```
# load data
data(sim_rs)

# log file for the best solution
cat(logging.file(sim_rs, 0))

# log file for the second solution
cat(logging.file(sim_rs, 2))

# log files for all solutions
cat(logging.file(sim_rs, NULL))
```

---

make.DemandPoints      *Generate demand points for RAP*

---

### Description

This function generates demand points to characterize a distribution of points.

### Usage

```
make.DemandPoints(
  points,
  n = 100L,
  quantile = 0.5,
  kernel.method = c("ks", "hypervolume")[1],
  ...
)
```

### Arguments

points	<a href="#">base::matrix()</a> object containing points.
n	integer number of demand points to use for each attribute space for each species. Defaults to 100L.
quantile	numeric quantile to generate demand points within. If 0 then demand points are generated across the full range of values the points intersect. Defaults to 0.5.
kernel.method	character name of kernel method to use to generate demand points. Defaults to 'ks'.
...	arguments passed to kernel density estimating functions

### Details

Broadly speaking, demand points are generated by fitting a kernel to the input points. A shape is then fit to the extent of the kernel, and then points are randomly generated inside the shape. The demand points are generated as random points inside the shape. The weights for each demand point are calculated the estimated density of input points at the demand point. By supplying 'ks' as an argument to method in kernel.method, the shape is defined using a minimum convex polygon [adehabitathR::mcp\(\)](#) and [ks::kde\(\)](#) is used to fit the kernel. Note this can only be used when the data is low-dimensional ( $d < 3$ ). By supplying "hypervolume" as an argument to method, the [hypervolume::hypervolume\(\)](#) function is used to create the demand points. This method can be used for hyper-dimensional data ( $d \ll 3$ ).

### Value

[DemandPoints\(\)](#) object.

### See Also

[hypervolume::hypervolume\(\)](#), [ks::kde\(\)](#), [adehabitathR::mcp\(\)](#).

**Examples**

```

# set random number generator seed
set.seed(500)

# load data
data(cs_spp, cs_space)

# generate species points
species.points <- randomPoints(cs_spp[[1]], n = 100, prob = TRUE)
env.points <- raster::extract(cs_space, species.points)

# generate demand points for a 1d space using ks
dps1 <- make.DemandPoints(points = env.points[, 1], kernel.method = "ks")

# print object
print(dps1)

## Not run:
# generate demand points for a 2d space using hypervolume
dps2 <- make.DemandPoints(points = env.points,
                          kernel.method = "hypervolume",
                          samples.per.point = 50,
                          verbose = FALSE)

# print object
print(dps2)

## End(Not run)

```

---

make.RapData

*Make data for RAP using minimal inputs*


---

**Description**

This function prepares spatially explicit planning unit, species data, and landscape data layers for RAP processing.

**Usage**

```

make.RapData(
  pus,
  species,
  spaces = NULL,
  amount.target = 0.2,
  space.target = 0.2,
  n.demand.points = 100L,
  kernel.method = c("ks", "hypervolume")[1],
  quantile = 0.5,
  species.points = NULL,

```

```

    n.species.points = ceiling(0.2 * raster::cellStats(species, "sum")),
    include.geographic.space = TRUE,
    scale = TRUE,
    verbose = FALSE,
    ...
)

```

## Arguments

pus	<a href="#">sp::SpatialPolygons()</a> with planning unit data.
species	<a href="#">raster::raster()</a> with species probability distribution data.
spaces	list of/or <a href="#">raster::raster()</a> representing projects of attribute space over geographic space. Use a list to denote separate attribute spaces.
amount.target	numeric vector for area targets (%) for each species. Defaults to 0.2 for each attribute space for each species.
space.target	numeric vector for attribute space targets (%) for each species. Defaults to 0.2 for each attribute space for each species and each space.
n.demand.points	integer number of demand points to use for each attribute space for each species. Defaults to 100L.
kernel.method	character name of kernel method to use to generate demand points. Use either "ks" or "hypervolume".
quantile	numeric quantile to generate demand points within. If species.points intersect. Defaults to 0.5.
species.points	list of/or <a href="#">sp::SpatialPointsDataFrame()</a> or <a href="#">sp::SpatialPoints()</a> with species presence records. Use a list of objects to represent different species. Must have the same number of elements as species. If not supplied then use n.species.points to sample points from the species distributions.
n.species.points	numeric vector specifying the number points to sample the species distributions to use to generate demand points. Defaults to 20% of the distribution.
include.geographic.space	logical should the geographic space be considered an attribute space?
scale	logical scale the attribute spaces to unit mean and standard deviation? This prevents overflow. Defaults to TRUE.
verbose	logical print statements during processing?
...	additional arguments to <a href="#">calcBoundaryData()</a> and <a href="#">calcSpeciesAverageInPus()</a> .

## See Also

[RapData](#), [RapData\(\)](#).

**Examples**

```
# load data
data(cs_pus, cs_spp, cs_space)
## Not run:
# make RapData object using the first 10 planning units in the dat
x <- make.RapData(cs_pus[1:10,], cs_spp, cs_space,
                 include.geographic.space = TRUE)
# print object
print(x)

## End(Not run)
```

---

ManualOpts

*Create ManualOpts object*

---

**Description**

This function creates a new ManualOpts object.

**Usage**

```
ManualOpts(NumberSolutions = 1L)
```

**Arguments**

NumberSolutions  
integer number of solutions to generate. Defaults to 1L.

**Value**

[ManualOpts\(\)](#) object

**See Also**

[ManualOpts](#).

**Examples**

```
# create ManualOpts object
ManualOpts(NumberSolutions = 1L)
```

---

ManualOpts-class	<i>ManualOpts: An S4 class to represent parameters for manually specified solutions</i>
------------------	---

---

### Description

This class is used to store parameters.

### Slots

NumberSolutions integer number of solutions.

### See Also

[ManualOpts\(\)](#).

---

maximum.targets	<i>Maximum targets</i>
-----------------	------------------------

---

### Description

This function accepts a [RapUnsolved\(\)](#) object and returns a data.frame containing the amount-based and space-based targets for each species and attribute space. These are calculated using a prioritization that contains all the available planning units. Note that the maximum amount-based targets are always 1.

### Usage

```
maximum.targets(x, verbose)

## S3 method for class 'RapUnsolvedOrSolved'
maximum.targets(x, verbose = FALSE)
```

### Arguments

x [RapUnsolved\(\)](#) or [RapSolved\(\)](#) object.

verbose logical should messages be printed during calculations? Defaults to FALSE.

### Value

data.frame object.

## Examples

```
# load RapSolved objects
data(sim_ru)

# calculate maximum metrics
maximum.targets(sim_ru)
```

---

names	<i>Names</i>
-------	--------------

---

## Description

This function sets or returns the species names in an object.

## Usage

```
## S3 replacement method for class 'RapData'
names(x) <- value

## S3 method for class 'RapData'
names(x)

## S3 replacement method for class 'RapUnsol0rSol1'
names(x) <- value

## S3 method for class 'RapUnsol0rSol1'
names(x)
```

## Arguments

x	<a href="#">RapData()</a> , <a href="#">RapUnsolved()</a> , or <a href="#">RapSolved()</a> object.
value	new species names.

## See Also

[RapData\(\)](#), [RapUnsolved\(\)](#), [RapSolved\(\)](#).

## Examples

```
# load data
data(sim_rs)

# show names
names(sim_rs)

# change names
names(sim_rs) <- c('spp1', 'spp2', 'spp3')
```

```
# show new names
names(sim_rs)
```

---

PlanningUnitPoints      *Create new PlanningUnitPoints object*

---

### Description

This function creates a new PlanningUnitPoints object.

### Usage

```
PlanningUnitPoints(coords, ids)
```

### Arguments

coords            `base::matrix()` coordinates for each point.  
ids                integer planning unit ids.

### See Also

[AttributeSpace](#).

### Examples

```
# create PlanningUnitPoints object
x <- PlanningUnitPoints(matrix(rnorm(150), ncol = 1), seq_len(150))

# print object
print(x)
```

---

PlanningUnitPoints-class

*PlanningUnitPoints: An S4 class to represent planning units in an attribute space*

---

### Description

This class is used to planning units in an attribute space.

### Slots

coords `base::matrix()` coordinates for each point.  
ids integer planning unit ids.

**See Also**

[AttributeSpace\(\)](#).

---

plot

*Plot object*

---

**Description**

This function plots the solutions contained in [RapSolved\(\)](#) objects. It can be used to show a single solution, or the the selection frequencies of planning units contained in a single [RapSolved\(\)](#) object. Additionally, two [RapSolved\(\)](#) objects can be supplied to plot the differences between them.

**Usage**

```
## S4 method for signature 'RapSolved,numeric'
plot(x, y, basemap = "none",
     pu.color.palette = c("#e5f5f9", "#00441b", "#FFFF00", "#FF0000"), alpha =
     ifelse(basemap == "none", 1, 0.7), grayscale = FALSE, main = NULL,
     force.reset = FALSE)
```

```
## S4 method for signature 'RapSolved,missing'
plot(x, y, basemap = "none",
     pu.color.palette = c("PuBu", "#FFFF00", "#FF0000"),
     alpha = ifelse(basemap == "none", 1, 0.7),
     grayscale = FALSE, main = NULL,
     force.reset = FALSE)
```

```
## S4 method for signature 'RapSolved,RapSolved'
plot(x, y, i = NULL, j = i,
     basemap = "none",
     pu.color.palette = ifelse(is.null(i), c("RdYlBu", "#FFFF00",
     "#FF0000"), "Accent"),
     alpha = ifelse(basemap == "none", 1, 0.7),
     grayscale = FALSE, main = NULL, force.reset = FALSE)
```

**Arguments**

x	<a href="#">RapSolved()</a> object.
y	Available inputs are: NULL to plot selection frequencies, numeric number to plot a specific solution, 0 to plot the best solution, and a <a href="#">RapSolved()</a> object to plot differences in solutions between objects. Defaults to NULL.
basemap	character object indicating the type of basemap to use (see <a href="#">basemap()</a> ). Valid options include "none", "roadmap", "mobile", "satellite", "terrain", "hybrid", "mapmaker-roadmap", "mapmaker-hybrid". Defaults to "none" such that no basemap is shown.

<code>pu.color.palette</code>	character vector of colors to indicate planning unit statuses. If plotting selection frequencies (i.e., <code>j = NULL</code> ), then defaults to a <code>c("PuBu", "#FFFF00", "#FF0000")</code> . Here, the first element corresponds to a color palette (per <code>RColorBrewer::brewer.pal()</code> ) and the last two elements indicate the colors for locked in and locked out planning units. Otherwise, the parameter defaults to a character vector of <code>c("grey30", "green", "yellow", "black", "gray80", "red", "orange")</code> .
<code>alpha</code>	numeric value to indicating the transparency level for coloring the planning units.
<code>grayscale</code>	logical should the basemap be gray-scaled?
<code>main</code>	character title for the plot. Defaults to <code>NULL</code> and a default title is used.
<code>force.reset</code>	logical if basemap data has been cached, should it be re-downloaded?
<code>i</code>	Available inputs are: <code>NULL</code> to plot selection frequencies. numeric to plot a specific solution, <code>0</code> to plot the best solution. This argument is only used when <code>y</code> is a <code>RapSolved()</code> object. Defaults to <code>NULL</code> .
<code>j</code>	Available inputs are: <code>NULL</code> to plot selection frequencies. numeric to plot a specific solution, <code>0</code> to plot the best solution. This argument is only used when <code>y</code> is a <code>RapSolved()</code> object. Defaults to argument <code>j</code> .

## Details

This function requires the **RgoogleMaps** package to be installed in order to create display a basemap.

## See Also

[RapSolved\(\)](#).

## Examples

```
# load example data set with solutions
data(sim_rs)

# plot selection frequencies
plot(sim_rs)

# plot best solution
plot(sim_rs, 0)

# plot second solution
plot(sim_rs, 2)

# plot different between best and second solutions
plot(sim_rs, sim_rs, 0, 2)
```

---

PolySet-class	<i>PolySet</i>
---------------	----------------

---

**Description**

Object contains PolySet data.

**See Also**

[PBSmapping::PolySet\(\)](#).

---

print	<i>Print objects</i>
-------	----------------------

---

**Description**

Prints objects.

**Usage**

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

## S3 method for class 'AttributeSpaces'
print(x, ..., header = TRUE)

## S3 method for class 'GurobiOpts'
print(x, ..., header = TRUE)

## S3 method for class 'ManualOpts'
print(x, ..., header = TRUE)

## S3 method for class 'RapData'
print(x, ..., header = TRUE)

## S3 method for class 'RapReliableOpts'
print(x, ..., header = TRUE)

## S3 method for class 'RapResults'
print(x, ..., header = TRUE)

## S3 method for class 'RapUnreliableOpts'
print(x, ..., header = TRUE)

## S3 method for class 'RapUnsolved'
```

```
print(x, ...)

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

### Arguments

x	<a href="#">GurobiOpts()</a> , <a href="#">RapUnreliableOpts()</a> , <a href="#">RapReliableOpts()</a> , <a href="#">RapData()</a> , <a href="#">RapUnsolved()</a> , <a href="#">RapResults()</a> , or <a href="#">RapSolved()</a> object.
...	not used.
header	logical should object header be included?

### See Also

[GurobiOpts\(\)](#), [RapUnreliableOpts\(\)](#), [RapReliableOpts\(\)](#), [RapData\(\)](#), [RapUnsolved\(\)](#), [RapResults\(\)](#), [RapSolved\(\)](#).

### Examples

```
# load data
data(sim_ru, sim_rs)

# print GurobiOpts object
print(GurobiOpts())

# print RapReliableOpts object
print(RapReliableOpts())

# print RapUnreliableOpts object
print(RapUnreliableOpts())

# print RapData object
print(sim_ru@data)

# print RapUnsolved object
print(sim_ru)

# print RapResults object
print(sim_rs@results)

# print RapSolved object
print(sim_rs)
```

---

prob.subset

*Subset probabilities above a threshold*

---

### Description

This function subsets out probabilities assigned to planning units above a threshold. It effectively sets the probability that species inhabit planning units to zero if they are below the threshold.

**Usage**

```

prob.subset(x, species, threshold)

## S3 method for class 'RapData'
prob.subset(x, species, threshold)

## S3 method for class 'RapUnsol0rSol1'
prob.subset(x, species, threshold)

```

**Arguments**

x	<a href="#">RapData()</a> , <a href="#">RapUnsolved()</a> , or <a href="#">RapSolved()</a> object.
species	integer vector specifying the index of the species to which the threshold should be applied.
threshold	numeric probability to use a threshold.

**Value**

[RapData\(\)](#) or [RapUnsolved\(\)](#) object depending on input object.

**See Also**

[RapData\(\)](#), [RapUnsolved\(\)](#), [RapSolved\(\)](#).

**Examples**

```

# load data
data(sim_ru)

# generate new object with first 10 planning units
sim_ru2 <- prob.subset(sim_ru, seq_len(3), c(0.1, 0.2, 0.3))

```

---

pu.subset

*Subset planning units*

---

**Description**

Subset planning units from a [RapData\(\)](#), [RapUnsolved\(\)](#), or [RapSolved\(\)](#) object.

**Usage**

```

pu.subset(x, pu)

## S3 method for class 'RapData'
pu.subset(x, pu)

## S3 method for class 'RapUnsol0rSol1'
pu.subset(x, pu)

```

**Arguments**

`x` [RapData\(\)](#), [RapUnsolved\(\)](#), or [RapSolved\(\)](#) object.  
`pu` integer vector to specify the index of planning units to subset.

**Value**

[RapData\(\)](#) or [RapUnsolved\(\)](#) object depending on input object.

**See Also**

[RapData\(\)](#), [RapUnsolved\(\)](#), [RapSolved\(\)](#).

**Examples**

```
# load data
data(sim_ru)

# generate new object with first 10 planning units
sim_ru2 <- pu.subset(sim_ru, seq_len(10))
```

---

randomPoints	<i>Sample random points from a RasterLayer</i>
--------------	--

---

**Description**

This function generates random points in a [raster::raster\(\)](#) object.

**Usage**

```
randomPoints(mask, n, prob = FALSE)
```

**Arguments**

`mask` [raster::raster\(\)](#) object  
`n` integer number of points to sample  
`prob` logical should the raster values be used as weights? Defaults to FALSE.

**Value**

[base::matrix\(\)](#) with x-coordinates, y-coordinates, and cell values.

**See Also**

This function is similar to [dismo::randomPoints](#).

## Examples

```
# simulate data
sim_pus <- sim.pus(225L)
sim_spp <- sim.species(sim_pus, model = "normal", n = 1, res = 0.25)

# generate points
pts1 <- randomPoints(sim_spp, n = 5)
pts2 <- randomPoints(sim_spp, n = 5, prob = TRUE)

# plot points
plot(sim_spp)
points(pts1, col = "red")
points(pts2, col = "black")
```

---

 rap

*Generate prioritizations using RAP*


---

## Description

This is a general function to create Rap objects from scratch and solve them to generate solutions.

## Usage

```
rap(
  pus,
  species,
  spaces = NULL,
  formulation = c("unreliable", "reliable")[1],
  solve = TRUE,
  ...
)
```

## Arguments

pus	<a href="#">sp::SpatialPolygons()</a> object representing planning units.
species	<a href="#">raster::raster()</a> object with species distribution data.
spaces	list of <a href="#">raster::raster()</a> objects. Each elements denotes the spatial distribution for each space. Defaults to NULL.
formulation	character to indicate if the "unreliable" or "reliable" formulation should be used to generate prioritizations. Defaults to "unreliable".
solve	logical should solutions be generated?
...	arguments are passed to <a href="#">GurobiOpts()</a> , <a href="#">make.RapData()</a> , and <a href="#">RapReliableOpts()</a> or <a href="#">RapUnreliableOpts()</a> functions.

**Value**

`RapSolved()` object if solve is TRUE, otherwise an `RapUnsolved()` is returned.

**Note**

Type `vignette("raptr")` to see the package vignette for a tutorial.

**See Also**

`GurobiOpts()`, `RapReliableOpts()`, `RapUnreliableOpts()` `RapData()`, `RapResults()`, `RapUnsolved()`, `RapSolved()`.

---

 RapData

*Create new RapData object*


---

**Description**

This function creates a "RapData" object using pre-processed data.

**Usage**

```
RapData(
  pu,
  species,
  targets,
  pu.species.probabilities,
  attribute.spaces,
  boundary,
  polygons = NA,
  skipchecks = FALSE,
  .cache = new.env()
)
```

**Arguments**

<code>pu</code>	<code>base::data.frame()</code> planning unit data. Columns must be "cost" (numeric), "area" (numeric), and "status" (integer).
<code>species</code>	<code>base::data.frame()</code> with species data. Columns must be "name" (character).
<code>targets</code>	<code>base::data.frame()</code> with species data. Columns must be "species" (integer), "target" (integer), "proportion" (numeric).
<code>pu.species.probabilities</code>	<code>base::data.frame()</code> with data on the probability of species in each planning unit. Columns must be "species", (integer), "pu" (integer), and "value" (numeric).
<code>attribute.spaces</code>	list of <code>AttributeSpaces()</code> objects with the demand points and planning unit coordinates.

boundary	<code>base::data.frame()</code> with data on the shared boundary length of planning units. Columns must be "id1" (integer), "id2" (integer), and "boundary" (integer).
polygons	<code>PBSmapping::PolySet()</code> planning unit spatial data or NULL if data not available.
skipchecks	logical Skip data integrity checks? May improve speed for big data sets.
.cache	<code>base::environment()</code> used to cache calculations.

**Value**

RapData object

**Note**

Generally, users are not encouraged to change arguments to `.cache`.

**See Also**

[PBSmapping::PolySet\(\)](#), [sp::SpatialPoints\(\)](#), [sp::SpatialPointsDataFrame\(\)](#), [make.RapData\(\)](#), [RapData](#).

**Examples**

```
# load data
data(cs_pus, cs_spp, cs_space)

# create data for RapData object
attribute.spaces <- list(
  AttributeSpaces(name = "geographic", list(
    AttributeSpace(
      planning.unit.points = PlanningUnitPoints(
        rgeos::gCentroid(cs_pus[1:10,], byid = TRUE)@coords, seq_len(10)),
      demand.points = make.DemandPoints(
        randomPoints(cs_spp[[1]], n = 10, prob = TRUE)),
      species = 1L))),
  AttributeSpaces(name = "environmental", list(
    AttributeSpace(
      planning.unit.points = PlanningUnitPoints(
        raster::extract(cs_space[[1]], cs_pus[1:10,], fun = mean),
        seq_len(10)),
      demand.points = make.DemandPoints(
        cs_space[[1]][raster::Which(!is.na(cs_space[[1]])]),
        species = 1L))))
pu.species.probabilities <- calcSpeciesAverageInPus(cs_pus[1:10,],
                                                  cs_spp[[1]])
polygons <- SpatialPolygons2PolySet(cs_pus[1:10,])
boundary <- calcBoundaryData(cs_pus[1:10,])

x <- RapData(pu = cs_pus@data[1:10,], species = data.frame(name = "test"),
            target = data.frame(species = 1L, target = 0:2,
                               proportion = 0.2),
```

```

        pu.species.proBABILITIES = pu.species.proBABILITIES,
        attribute.spaces = attribute.spaces,
        polygons = polygons,
        boundary = boundary)

# print object
print(x)

```

---

 RapData-class

*RapData: An S4 class to represent RAP input data*


---

### Description

This class is used to store RAP input data.

### Slots

polygons [PBSmapping::PolySet\(\)](#) planning unit spatial data or NULL if data not available.

pu [base::data.frame\(\)](#) planning unit data. Columns must be "cost" (numeric), "area" (numeric), and "status" (integer).

species [base::data.frame\(\)](#) with species data. Columns must be "name" (character).

targets [base::data.frame\(\)](#) with species data. Columns must be "species" (integer), "target" (integer), "proportion" (numeric).

pu.species.proBABILITIES [base::data.frame\(\)](#) with data on the probability of species in each planning unit. Columns must be "species" (integer), "pu" (integer), and "value" (numeric).

attribute.spaces list of [AttributeSpaces](#) objects with the demand points and planning unit coordinates.

boundary [base::data.frame\(\)](#) with data on the shared boundary length of planning units. Columns must be "id1" (integer), "id2" (integer), and "boundary" (numeric).

skipchecks logical Skip data integrity checks? May improve speed for big data sets.

.cache [base::environment\(\)](#) used to cache calculations.

### See Also

[PBSmapping::PolySet\(\)](#).

---

 RapOpts-class

*RapOpts class*


---

### Description

Object is either [RapReliableOpts\(\)](#) or [RapUnreliableOpts\(\)](#).

---

RapReliableOpts      *Create RapReliableOpts object*

---

### Description

This function creates a new RapReliableOpts object.

### Usage

```
RapReliableOpts(BLM = 0, failure.multiplier = 1.1, max.r.level = 5L)
```

### Arguments

BLM                    numeric boundary length modifier. Defaults to 0.  
failure.multiplier    numeric multiplier for failure planning unit. Defaults to 1.1.  
max.r.level            numeric maximum R failure level for approximation. Defaults to 5L.

### Value

RapReliableOpts object

### See Also

[RapReliableOpts](#).

### Examples

```
# create RapReliableOpts using defaults
RapReliableOpts(BLM = 0, failure.multiplier = 1.1, max.r.level = 5L)
```

---

RapReliableOpts-class    *RapReliableOpts: An S4 class to represent input parameters for the reliable formulation of RAP.*

---

### Description

This class is used to store input parameters for the reliable formulation of RAP.

### Slots

BLM numeric boundary length modifier. Defaults to 0.  
failure.multiplier numeric multiplier for failure planning unit. Defaults to 1.1.  
max.r.level numeric maximum R failure level for approximation. Defaults to 5L.

**See Also**

[RapReliableOpts\(\)](#).

---

 RapResults

 Create RapResults object
 

---

**Description**

This function creates a new [RapResults\(\)](#) object.

**Usage**

```
RapResults(
  summary,
  selections,
  amount.held,
  space.held,
  logging.file,
  .cache = new.env()
)
```

**Arguments**

summary	<a href="#">base::data.frame()</a> with summary information on solutions. See details below for more information.
selections	<a href="#">base::matrix()</a> with binary selections. The cell $x_{ij}$ denotes if planning unit $j$ is selected in the $i$ 'th solution.
amount.held	<a href="#">base::matrix()</a> with the amount held for each species in each solution.
space.held	<a href="#">base::matrix()</a> with the proportion of attribute space sampled for each species in each solution.
logging.file	character Gurobi log files.
.cache	<a href="#">base::environment()</a> used to cache calculations.

**Details**

The summary table follows Marxan conventions (<https://marxansolutions.org/>). The columns are:

**Run\_Number** The index of each solution in the object.

**Status** The status of the solution. The values in this column correspond to outputs from the Gurobi software package ([https://www.gurobi.com/documentation/6.5/refman/optimization\\_status\\_codes.html](https://www.gurobi.com/documentation/6.5/refman/optimization_status_codes.html)).

**Score** The objective function for the solution.

**Cost** Total cost associated with a solution.

- Planning\_Units** Number of planning units selected in a solution.
- Connectivity\_Total** The total amount of shared boundary length between all planning units. All solutions in the same object should have equal values for this column.
- Connectivity\_In** The amount of shared boundary length among planning units selected in the solution.
- Connectivity\_Edge** The amount of exposed boundary length in the solution.
- Connectivity\_Out** The number of shared boundary length among planning units not selected in the solution.
- Connectivity\_Fraction** The ratio of shared boundary length in the solution (`Connectivity_In`) to the total amount of boundary length (`Connectivity_Edge`). This ratio is an indicator of solution quality. Solutions with a lower ratio will have less planning units and will be more efficient.

**Value**

RapResults object

**Note**

slot `best` is automatically determined based on data in `summary`.

**See Also**

[RapResults](#) `read.RapResults()`.

---

RapResults-class

*RapResults: An S4 class to represent RAP results*

---

**Description**

This class is used to store RAP results.

**Details**

The summary table follows Marxan conventions (<https://marxansolutions.org/>). The columns are:

- Run\_Number** The index of each solution in the object.
- Status** The status of the solution. The values in this column correspond to outputs from the Gurobi software package ([https://www.gurobi.com/documentation/6.5/refman/optimization\\_status\\_codes.html](https://www.gurobi.com/documentation/6.5/refman/optimization_status_codes.html)).
- Score** The objective function for the solution.
- Cost** Total cost associated with a solution.
- Planning\_Units** Number of planning units selected in a solution.

**Connectivity\_Total** The total amount of shared boundary length between all planning units. All solutions in the same object should have equal values for this column.

**Connectivity\_In** The amount of shared boundary length among planning units selected in the solution.

**Connectivity\_Edge** The amount of exposed boundary length in the solution.

**Connectivity\_Out** The number of shared boundary length among planning units not selected in the solution.

**Connectivity\_Fraction** The ratio of shared boundary length in the solution (`Connectivity_In`) to the total amount of boundary length (`Connectivity_Edge`). This ratio is an indicator of solution quality. Solutions with a lower ratio will have less planning units and will be more efficient.

### Slots

summary `base::data.frame()` with summary information on solutions.

selections `base::matrix()` with binary selections. The cell  $x_{ij}$  denotes if planning unit  $j$  is selected in the  $i$ 'th solution.

amount.held `base::matrix()` with the amount held for each species in each solution.

space.held `base::matrix()` with the proportion of attribute space sampled for each species in each solution.

best integer with index of best solution.

logging.file character Gurobi log files.

.cache `base::environment()` used to store extra data.

### See Also

[RapResults\(\)](#), [read.RapResults\(\)](#).

---

RapSolved

*Create new RapSolved object*

---

### Description

This function creates a [RapSolved\(\)](#) object.

### Usage

```
RapSolved(unsolved, solver, results)
```

### Arguments

unsolved [RapUnsolved\(\)](#) object.

solver [GurobiOpts\(\)](#) or [ManualOpts\(\)](#) object.

results [RapResults\(\)](#) object.

**Value**

`RapSolved()` object.

**See Also**

`RapSolved`, `RapResults`, `link{solve}`.

---

`RapSolved`-class

*RapSolved: An S4 class to represent RAP inputs and outputs*

---

**Description**

This class is used to store RAP input and output data in addition to input parameters.

**Slots**

opts `RapReliableOpts()` or `RapUnreliableOpts()` object used to store input parameters.

solver `GurobiOpts()` or `ManualOpts()` object used to store solver information/parameters.

data `RapData()` object used to store input data.

results `RapResults()` object used to store results.

**See Also**

`RapReliableOpts`, `RapUnreliableOpts`, `RapData`, `RapResults`.

---

raptr

*raptr: Representative and Adequate Prioritization Toolkit in R*

---

**Description**

Biodiversity is in crisis. The overarching aim of conservation is to preserve biodiversity patterns and processes. To this end, protected areas are established to buffer species and preserve biodiversity processes. But resources are limited and so protected areas must be cost-effective. This package contains tools to generate plans for protected areas (prioritizations). Conservation planning data are used to construct an optimization problem, which is then solved to yield prioritizations. To solve the optimization problems in a feasible amount of time, this package uses the commercial 'Gurobi' software package (obtained from <https://www.gurobi.com/>). For more information on using this package, see Hanson et al. (2018).

## Details

The main classes used in this package are used to store input data and prioritizations:

**GurobiOpts** parameters for solving optimization problems using Gurobi.

**RapReliableOpts** parameters for the reliable formulation of RAP.

**RapUnreliableOpts** parameters for the unreliable formulation of RAP.

**RapData** planning unit, species data, and demand points for RAP.

**RapUnsolved** contains all the data and input parameters required to generate prioritizations using RAP. This class contains a **GurobiOpts** object, a **RapReliableOpts** or **RapUnreliableOpts** object, and a **RapData** object.

**RapResults** prioritizations and summary statistics on their performance.

**RapSolved** contains all the input data, parameters and output data. This class contains all the objects in a **RapUnsolved()** object and also a **RapResults** object.

Type `vignette("raptr")` for a tutorial on how to use this package.

## References

Hanson JO, Rhodes JR, Possingham HP & Fuller RA (2018) raptr: Representative and Adequate Prioritization", Toolkit in R. *Methods in Ecology & Evolution*, **9**: 320–330. DOI: 10.1111/2041-210X.12862.

---

RapUnreliableOpts      *Create RapUnreliableOpts object*

---

## Description

This function creates a new RapUnreliableOpts object.

## Usage

```
RapUnreliableOpts(BLM = 0)
```

## Arguments

BLM                      numeric boundary length modifier. Defaults to 0.

## Value

**RapUnreliableOpts()** object

## See Also

[RapUnreliableOpts](#).

**Examples**

```
# create RapUnreliableOpts using defaults
RapUnreliableOpts(BLM = 0)
```

---

RapUnreliableOpts-class

*RapUnreliableOpts: An S4 class to represent parameters for the unreliable RAP problem*

---

**Description**

This class is used to store input parameters for the unreliable RAP problem formulation.

**Slots**

BLM numeric boundary length modifier. Defaults to 0.

---

RapUnsolved

*Create a new RapUnsolved object*

---

**Description**

This function creates a [RapUnsolved\(\)](#) object using a [GurobiOpts\(\)](#), a [RapReliableOpts\(\)](#) or [RapUnreliableOpts\(\)](#) object, and a [RapData\(\)](#) object.

**Usage**

```
RapUnsolved(opts, data)
```

**Arguments**

opts            [RapReliableOpts\(\)](#) or [RapUnreliableOpts\(\)](#) object.

data            [RapData\(\)](#) object.

**Value**

[RapUnsolved\(\)](#) object.

**See Also**

[RapReliableOpts](#), [RapUnreliableOpts](#), [RapData](#).

**Examples**

```
## Not run:
# set random number generator seed
set.seed(500)

# load data
data(cs_pus, cs_spp)

# create inputs for RapUnsolved
ro <- RapUnreliableOpts()
rd <- make.RapData(cs_pus[seq_len(10), ], cs_spp, NULL,
                  include.geographic.space = TRUE, n.demand.points = 5L)

# create RapUnsolved object
ru <- RapUnsolved(ro, rd)

# print object
print(ru)

## End(Not run)
```

---

RapUnsolved-class

*RapUnsolved: An S4 class to represent RAP inputs*

---

**Description**

This class is used to store RAP input data and input parameters.

**Slots**

opts [RapReliableOpts\(\)](#) or [RapUnreliableOpts\(\)](#) object used to store input parameters.

data [RapData\(\)](#) object used to store input data.

**See Also**

[RapReliableOpts](#), [RapUnreliableOpts](#), [RapData](#).

---

`rrap.proportion.held`

*Proportion held using reliable RAP formulation.*

---

**Description**

This is a convenience function to quickly calculate the proportion of variation that one set of points captures in another set of points using the reliable formulation.

**Usage**

```
rrap.proportion.held(  
  pu.coordinates,  
  pu.probabilities,  
  dp.coordinates,  
  dp.weights,  
  failure.distance,  
  maximum.r.level = as.integer(length(pu.probabilities))  
)
```

**Arguments**

`pu.coordinates` [base::matrix\(\)](#) of planning unit coordinates.  
`pu.probabilities` numeric vector of planning unit probabilities.  
`dp.coordinates` [base::matrix\(\)](#) of demand point coordinates.  
`dp.weights` numeric vector of demand point weights.  
`failure.distance` numeric indicating the cost of the failure planning unit.  
`maximum.r.level` integer maximum failure (R) level to use for calculations.

**Value**

numeric value indicating the proportion of variation that the demand points explain in the planning units

**Examples**

```
rrap.proportion.held(as.matrix(iris[1:2,-5]), runif(1:2),  
                    as.matrix(iris[1:5,-5]), runif(1:5), 10)
```

---

score

*Solution score*

---

**Description**

Extract solution score from [RapResults\(\)](#) or [RapSolved\(\)](#) object.

**Usage**

```
score(x, y)

## S3 method for class 'RapResults'
score(x, y = 0)

## S3 method for class 'RapSolved'
score(x, y = 0)
```

**Arguments**

x [RapResults\(\)](#) or [RapSolved\(\)](#) object.

y Available inputs include: NULL to return all scores, integer number specifying the solution for which the score should be returned, and 0 to return score for the best solution.

**Value**

matrix or numeric vector with solution score(s) depending on arguments.

**See Also**

[RapResults\(\)](#), [RapSolved\(\)](#).

**Examples**

```
# load data
data(sim_rs)

# score for the best solution
score(sim_rs, 0)

# score for the second solution
score(sim_rs, 2)

# score for all solutions
score(sim_rs, NULL)
```

---

selections

*Extract solution selections*

---

**Description**

Extract selections for a given solution from a [RapResults\(\)](#) or [RapSolved\(\)](#) object.

**Usage**

```
selections(x, y)

## S3 method for class 'RapResults'
selections(x, y = 0)

## S3 method for class 'RapSolved'
selections(x, y = 0)
```

**Arguments**

x                    [RapResults\(\)](#) or [RapSolved\(\)](#) object.

y                    NULL to return all values, integer 0 to return values for the best solution, integer value greater than 0 for y'th solution value.

**Value**

[base::matrix\(\)](#) or numeric vector depending on arguments.

**See Also**

[RapResults\(\)](#), [RapSolved\(\)](#).

**Examples**

```
# load data
data(sim_rs)

# selections for the best solution
selections(sim_rs, 0)

# selections for the second solution
selections(sim_rs, 2)

# selections for each solution
selections(sim_rs)
```

---

show

*Show objects*

---

**Description**

Shows objects.

**Usage**

```
## S4 method for signature 'GurobiOpts'  
show(object)  
  
## S4 method for signature 'ManualOpts'  
show(object)  
  
## S4 method for signature 'RapData'  
show(object)  
  
## S4 method for signature 'RapReliableOpts'  
show(object)  
  
## S4 method for signature 'RapResults'  
show(object)  
  
## S4 method for signature 'RapUnreliableOpts'  
show(object)  
  
## S4 method for signature 'RapUnsolved'  
show(object)  
  
## S4 method for signature 'RapSolved'  
show(object)
```

**Arguments**

object            [GurobiOpts\(\)](#), [RapUnreliableOpts\(\)](#), [RapReliableOpts\(\)](#), [RapData\(\)](#), [RapUnsolved\(\)](#), [RapResults\(\)](#), or [RapSolved\(\)](#) object.

**See Also**

[GurobiOpts\(\)](#), [RapUnreliableOpts\(\)](#), [RapReliableOpts\(\)](#), [RapData\(\)](#), [RapUnsolved\(\)](#), [RapResults\(\)](#), [RapSolved\(\)](#).

**Examples**

```
# load data  
data(sim_ru, sim_rs)  
  
# show GurobiOpts object  
GurobiOpts()  
  
# show RapReliableOpts object  
RapReliableOpts()  
  
# show RapUnreliableOpts object  
RapUnreliableOpts()  
  
# show RapData object
```

```
sim_ru@data

# show RapUnsolved object
sim_ru

# show RapResults object
sim_rs@results

# show RapSolved object
sim_rs
```

---

sim.pus	<i>Simulate planning units</i>
---------	--------------------------------

---

## Description

This function simulates planning units for RAP.

## Usage

```
sim.pus(  
  n,  
  xmn = -sqrt(n)/2,  
  xmx = sqrt(n)/2,  
  ymn = -sqrt(n)/2,  
  ymx = sqrt(n)/2  
)
```

## Arguments

n	integer number of planning units. $\sqrt{n}$ must yield a valid number.
xmn	numeric value for minimum x-coordinate.
xmx	numeric value for maximum x-coordinate.
ymn	numeric value for minimum y-coordinate.
ymx	numeric value for maximum y-coordinate.

## Details

Square planning units are generated in the shape of a square. Default coordinate arguments are such that the planning units will be centered at origin. The data slot contains an "id" (integer), "cost" (numeric), "status" (integer), and "area" (numeric).

## Value

`sp::SpatialPolygons()` with planning units.

**Examples**

```
# generate 225 square planning units arranged in a square
# with 1 unit height / width
x <- sim.pus(225)

# generate 225 rectangular pus arranged in a square
y <- sim.pus(225, xmn = -5, xmx = 10, ymn = -5, ymx = 5)
## Not run:
par(mfrow = c(1, 2))
plot(x, main = "x")
plot(y, main = "y")
par(mfrow = c(1, 1))

## End(Not run)
```

---

sim.space

*Simulate attribute space data for RAP*


---

**Description**

This function simulates attribute space data for RAP.

**Usage**

```
sim.space(x, ...)
```

## S3 method for class 'RasterLayer'

```
sim.space(x, d = 2, model = 0.2, ...)
```

## S3 method for class 'SpatialPolygons'

```
sim.space(x, res, d = 2, model = 0.2, ...)
```

**Arguments**

x	<code>raster::raster()</code> or <code>sp::SpatialPolygons()</code> object delineate the spatial extent to delineate study area.
...	not used.
d	integer number of dimensions. Defaults to 2.
model	numeric scale parameter for simulating spatially auto-correlated data using Gaussian random fields. Higher values produce patchier data with more well defined clusters, and lower values produce more evenly distributed data. Defaults to 0.2.
res	numeric resolution to simulate distributions. Only needed when <code>sp::SpatialPolygons()</code> are supplied.

**Value**

`raster::stack()` with layers for each dimension of the space.

## Examples

```
# simulate planning units
sim_pus <- sim.pus(225L)

# simulate 1d space using RasterLayer
s1 <- sim.space(blank.raster(sim_pus, 1), d = 1)

# simulate 1d space using SpatialPolygons
s2 <- sim.space(sim_pus, res = 1, d = 1)

# simulate 2d space using SpatialPolygons
s3 <- sim.space(sim_pus, res = 1, d = 2)

# plot simulated spaces
par(mfrow = c(2,2))
plot(s1, main = "s1")
plot(s2, main = "s2")
plot(s3[[1]], main = "s3: first dimension")
plot(s3[[2]], main = "s3: second dimension")
```

---

sim.species

*Simulate species distribution data for RAP*

---

## Description

This function simulates species distributions for RAP.

## Usage

```
sim.species(x, ...)
```

```
## S3 method for class 'RasterLayer'
sim.species(x, n = 1, model = "normal", ...)
```

```
## S3 method for class 'SpatialPolygons'
sim.species(x, res, n = 1, model = "normal", ...)
```

## Arguments

x	<code>raster::raster()</code> or <code>sp::SpatialPolygons()</code> object delineate the spatial extent to delineate study area.
...	not used.
n	integer number of species. Defaults to 1.
model	character or numeric for simulating data. If a character value is supplied, then the following values can be used to simulate species distributions with particular characteristics: "uniform", "normal", and "bimodal". If a numeric

value is supplied, then this is used to simulate species distributions using a Gaussian random field, where the numeric value is treated as the scale parameter. Defaults to "normal".

`res` numeric resolution to simulate distributions. Only needed when `sp::SpatialPolygons()` are supplied.

### Value

`raster::stack()` with layers for each species.

### Examples

```
# make polygons
sim_pus <- sim.pus(225L)

# simulate 1 uniform species distribution using RasterLayer
s1 <- sim.species(blank.raster(sim_pus, 1), n = 1, model = "uniform")

# simulate 1 uniform species distribution based on SpatialPolygons
s2 <- sim.species(sim_pus, res = 1, n = 1, model = "uniform")

# simulate 1 normal species distributions
s3 <- sim.species(sim_pus, res = 1, n = 1, model = "normal")

# simulate 1 bimodal species distribution
s4 <- sim.species(sim_pus, res = 1, n = 1, model = "bimodal")

# simulate 1 species distribution using a random field
s5 <- sim.species(sim_pus, res = 1, n = 1, model = 0.2)

# plot simulations
par(mfrow = c(2,2))
plot(s2, main = "constant")
plot(s3, main = "normal")
plot(s4, main = "bimodal")
plot(s5, main = "random field")
```

---

simulated\_data

*Simulated dataset for a conservation planning exercise*

---

### Description

This dataset contains all the data needed to generate prioritizations for three simulated species. This dataset contains planning units, species distribution maps, and demand points for each species. For the purposes of exploring the behaviour of the problem, demand points were generated using the centroids of planning units and the probability that they are occupied by the species. Note that methodology is not encouraged for real-world conservation planning.

**Format**

**sim\_ru** `RapUnsolved()` object with all the simulated data.

**sim\_rs** `RapSolved()` object with 5 near-optimal solutions.

**Details**

The species were simulated to represent various simplified species distributions.

**uniform** This species has an equal probability (0.5) of occurring in all planning units.

**normal** This species has a single range-core where it is most likely to be found. It is less likely to be found in areas further away from the center of its range.

**bimodal** This species has two distinct ecotypes. Each ecotype has its own core and marginal area.

**Examples**

```
# load data
data(sim_ru, sim_rs)

# plot species distributions
spp.plot(sim_ru, 1)
spp.plot(sim_ru, 2)
spp.plot(sim_ru, 3)

# plot selection frequencies
plot(sim_rs)

# plot best solution
plot(sim_rs, 0)
```

---

solve

*Solve RAP object*

---

**Description**

This function uses Gurobi to find prioritizations using the input parameter and data stored in a `RapUnsolved()` object, and returns a `RapSolved()` object with outputs in it.

**Usage**

```
## S4 method for signature 'RapUnsolOrSol,missing'
solve(a, b, ..., verbose = FALSE)

## S4 method for signature 'RapUnsolOrSol,GurobiOpts'
solve(a, b, verbose = FALSE)

## S4 method for signature 'RapUnsolOrSol,matrix'
solve(a, b, verbose = FALSE)
```

```
## S4 method for signature 'RapUnsolOrSol,numeric'
solve(a, b, verbose = FALSE)

## S4 method for signature 'RapUnsolOrSol,logical'
solve(a, b, verbose = FALSE)
```

### Arguments

a	<a href="#">RapUnsolved()</a> or <a href="#">RapSolved()</a> object.
b	missing to generate solutions using Gurobi. Prioritizations can be specified using <code>logical</code> , <code>numeric</code> , or <code>base::matrix()</code> objects. This may be useful for evaluating the performance of solutions obtained using other software.
...	not used.
verbose	logical should messages be printed during creation of the initial model matrix?.

### Value

[RapSolved\(\)](#) object

### Note

This function is used to solve a [RapUnsolved\(\)](#) object that has all of its inputs generated. The `rap` function (without lower case 'r') provides a more general interface for generating inputs and outputs.

### See Also

[RapUnsolved\(\)](#), [RapSolved\(\)](#).

### Examples

```
# load RapUnsolved object
data(sim_ru)
## Not run:
# solve it using Gurobi
sim_rs <- solve(sim_ru)

# evaluate manually specified solution using planning unit indices
sim_rs2 <- solve(sim_ru, seq_len(10))

# evaluate manually specified solution using binary selections
sim_rs3 <- solve(sim_ru, c(rep(TRUE, 10), rep(FALSE, 90)))

# evaluate multiple manually specified solutions
sim_rs4 <- solve(sim_ru, matrix(sample(c(0, 1), size = 500, replace = TRUE),
                                ncol = 100, nrow = 5))

## End(Not run)
```

---

SolverOpts-class	<i>SolverOpts class</i>
------------------	-------------------------

---

**Description**

Object stores parameters used to solve problems.

**See Also**

[GurobiOpts\(\)](#).

---

space.held	<i>Extract attribute space held for a solution</i>
------------	--

---

**Description**

This function returns the attribute space held for each species in a solution.

**Usage**

```
space.held(x, y, species, space)
```

```
## S3 method for class 'RapSolved'
space.held(x, y = 0, species = NULL, space = NULL)
```

**Arguments**

x	<a href="#">RapResults()</a> or <a href="#">RapSolved()</a> object.
y	Available inputs include: NULL to return all values, integer number specifying the solution for which the value should be returned, and 0 to return the value for the best solution.
species	NULL for all species or integer indicating species.
space	NULL for all spaces or integer indicating a specific space.

**Value**

codematrix object.

**See Also**

[RapResults\(\)](#), [RapSolved\(\)](#).

**Examples**

```
# load data
data(sim_rs)

# space held (%) for each species in best solution
space.held(sim_rs, 0)

# space held (%) for each species in second solution
space.held(sim_rs, 2)

# space held (%) for each species in each solution
space.held(sim_rs)
```

---

space.plot

*Plot space*


---

**Description**

This function plots the distribution of planning units and the distribution of demand points for a particular species in an attribute space. Note that this function only works for attribute spaces with one, two, or three dimensions.

**Usage**

```
space.plot(x, species, space, ...)

## S3 method for class 'RapData'
space.plot(
  x,
  species,
  space = 1,
  pu.color.palette = c("#4D4D4D4D", "#00FF0080", "#FFFF0080", "#FF00004D"),
  main = NULL,
  ...
)

## S3 method for class 'RapUnsolved'
space.plot(
  x,
  species,
  space = 1,
  pu.color.palette = c("#4D4D4D4D", "#00FF0080", "#FFFF0080", "#FF00004D"),
  main = NULL,
  ...
)
```

```
## S3 method for class 'RapSolved'
space.plot(
  x,
  species,
  space = 1,
  y = 0,
  pu.color.palette = c("#4D4D4D", "#00FF0080", "#FFFF0080", "#FF00004D"),
  main = NULL,
  ...
)
```

### Arguments

x	<a href="#">RapData()</a> , <a href="#">RapUnsolved()</a> , or <a href="#">RapSolved()</a> object.
species	character name of species, or integer index for species.
space	integer index of attribute space.
...	not used.
pu.color.palette	character vector of colors indicate planning unit statuses. Defaults to c("grey30", "green", "black", "red") which indicate not selected, selected, locked in, and locked out (respectively).
main	character title for the plot. Defaults to NULL and a default title is used.
y	integer number specifying the solution to be plotted. The value 0 can be used to plot the best solution.

### Examples

```
# load RapSolved objects
data(sim_ru, sim_rs)

# plot first species in first attribute space
space.plot(sim_ru, 1, 1)

# plot distribution of solutions for first species in first attribute space
space.plot(sim_rs, 1, 1)
```

---

space.target

*Attribute space targets*

---

### Description

This function sets or returns the attribute space targets for each species.

**Usage**

```
space.target(x, species, space)

space.target(x, species, space) <- value

## S3 method for class 'RapData'
space.target(x, species = NULL, space = NULL)

## S3 replacement method for class 'RapData'
space.target(x, species = NULL, space = NULL) <- value

## S3 method for class 'RapUnsolOrSol'
space.target(x, species = NULL, space = NULL)

## S3 replacement method for class 'RapUnsolOrSol'
space.target(x, species = NULL, space = NULL) <- value
```

**Arguments**

x	<a href="#">RapData()</a> , <a href="#">RapUnsolved()</a> , or <a href="#">RapSolved()</a> object.
species	NULL for all species or integer indicating species.
space	NULL for all spaces or integer indicating a specific space.
value	numeric new target.

**Value**

numeric matrix.

**See Also**

[RapData\(\)](#), [RapResults\(\)](#), [RapSolved\(\)](#).

**Examples**

```
# load data
data(sim_rs)

# extract space targets for all species
space.target(sim_rs)

# set space targets for all species
space.target(sim_rs) <- 0.1

# extract target for first species for first space
space.target(sim_rs, 1, 1)

# set space targets for first species for first space
space.target(sim_rs, 1, 1) <- 0.5
```

---

`SpatialPolygons2PolySet`*Convert SpatialPolygons to PolySet data*

---

### Description

This function converts spatial `sp::SpatialPolygons()` and `sp::SpatialPolygonsDataFrame()` objects to `PBSmapping::PolySet()` objects.

### Usage

```
SpatialPolygons2PolySet(x, n_preallocate)

## S3 method for class 'SpatialPolygonsDataFrame'
SpatialPolygons2PolySet(x, n_preallocate = 10000L)

## S3 method for class 'SpatialPolygons'
SpatialPolygons2PolySet(x, n_preallocate = 10000L)
```

### Arguments

`x` `sp::SpatialPolygons()` or `sp::SpatialPolygonsDataFrame()` object.  
`n_preallocate` integer How much memory should be preallocated for processing? Ideally, this number should equal the number of vertices in the `sp::SpatialPolygons()` object. If data processing is taking too long consider increasing this value.

### Value

`PBSmapping::PolySet()` object.

### Note

Be aware that this function is designed to be as fast as possible, but as a result it depends on C++ code and if used inappropriately this function will crash R.

### See Also

For a slower, more stable equivalent see `maptools::SpatialPolygons2PolySet`.

### Examples

```
# generate SpatialPolygons object
sim_pus <- sim.pus(225L)

# convert to PolySet
x <- SpatialPolygons2PolySet(sim_pus)
```

---

spp.plot	<i>Plot species</i>
----------	---------------------

---

### Description

This function plots the distribution of species across the study area.

### Usage

```
spp.plot(x, species, ...)

## S3 method for class 'RapData'
spp.plot(
  x,
  species,
  prob.color.palette = "YlGnBu",
  pu.color.palette = c("#4D4D4D", "#00FF00", "#FFFF00", "#FF0000"),
  basemap = "none",
  alpha = ifelse(identical(basemap, "none"), 1, 0.7),
  grayscale = FALSE,
  main = NULL,
  force.reset = FALSE,
  ...
)

## S3 method for class 'RapUnsolved'
spp.plot(
  x,
  species,
  prob.color.palette = "YlGnBu",
  pu.color.palette = c("#4D4D4D", "#00FF00", "#FFFF00", "#FF0000"),
  basemap = "none",
  alpha = ifelse(basemap == "none", 1, 0.7),
  grayscale = FALSE,
  main = NULL,
  force.reset = FALSE,
  ...
)

## S3 method for class 'RapSolved'
spp.plot(
  x,
  species,
  y = 0,
  prob.color.palette = "YlGnBu",
  pu.color.palette = c("#4D4D4D", "#00FF00", "#FFFF00", "#FF0000"),
  basemap = "none",
```

```

alpha = ifelse(basemap == "none", 1, 0.7),
grayscale = FALSE,
main = NULL,
force.reset = FALSE,
...
)

```

## Arguments

x	<a href="#">RapData()</a> , <a href="#">RapUnsolved()</a> , or <a href="#">RapSolved()</a> object.
species	character name of species, or integer index for species.
...	not used.
prob.color.palette	character name of color palette to denote probability of occupancy of the species in planning units (see <a href="#">RColorBrewer::brewer.pal()</a> ). Defaults to "YlGnBu".
pu.color.palette	character vector of colors to indicate planning unit statuses. Defaults to c("grey30", "green", "black", "red") which indicate not selected, selected, locked in, and locked out (respectively).
basemap	character object indicating the type of basemap to use (see <a href="#">basemap()</a> ). Valid options include "none", "roadmap", "mobile", "satellite", "terrain", "hybrid", "mapmaker-roadmap", "mapmaker-hybrid". Defaults to "none" such that no basemap is shown.
alpha	numeric value to indicating the transparency level for coloring the planning units.
grayscale	logical should the basemap be gray-scaled?
main	character title for the plot. Defaults to NULL and a default title is used.
force.reset	logical if basemap data has been cached, should it be re-downloaded?
y	NULL integer 0 to return values for the best solution, integer value greater than 0 for y'th solution value.

## Details

This function requires the **RgoogleMaps** package to be installed in order to create display a basemap.

## Examples

```

# load RapSolved objects
data(sim_ru, sim_rs)

# plot first species in sim_ru
spp.plot(sim_ru, species = 1)

# plot "bimodal" species in sim_rs
spp.plot(sim_rs, species = "bimodal")

```

---

spp.subset	<i>Subset species</i>
------------	-----------------------

---

### Description

Subset species from a [RapData\(\)](#), [RapUnsolved\(\)](#), or [RapSolved\(\)](#) object.

### Usage

```
spp.subset(x, species)

## S3 method for class 'RapData'
spp.subset(x, species)

## S3 method for class 'RapUnsolvedOrSolved'
spp.subset(x, species)
```

### Arguments

**x** [RapData\(\)](#), [RapUnsolved\(\)](#), or [RapSolved\(\)](#) object.  
**species** integer, or character vectors to specify the index or species names to subset.

### Value

[RapData\(\)](#) or [RapUnsolved\(\)](#) object depending on input object.

### See Also

[RapData\(\)](#), [RapUnsolved\(\)](#), [RapSolved\(\)](#).

### Examples

```
# load data
data(sim_ru)

# generate new object with only species 1
sim_ru2 <- spp.subset(sim_ru, 1)
```

---

summary	<i>Summary of solutions</i>
---------	-----------------------------

---

### Description

Extracts summary of solutions in a `RapResults()` or `RapSolved()` object.

### Arguments

object	<code>RapResults()</code> , or <code>RapSolved()</code> object.
...	not used.

### Details

This table follows Marxan conventions (<https://marxansolutions.org/>). The columns are:

**Run\_Number** The index of each solution in the object.

**Status** The status of the solution. The values in this column correspond to outputs from the Gurobi software package ([https://www.gurobi.com/documentation/6.5/refman/optimization\\_status\\_codes.html](https://www.gurobi.com/documentation/6.5/refman/optimization_status_codes.html)).

**Score** The objective function for the solution.

**Cost** Total cost associated with a solution.

**Planning\_Units** Number of planning units selected in a solution.

**Connectivity\_Total** The total amount of shared boundary length between all planning units. All solutions in the same object should have equal values for this column.

**Connectivity\_In** The amount of shared boundary length among planning units selected in the solution.

**Connectivity\_Edge** The amount of exposed boundary length in the solution.

**Connectivity\_Out** The number of shared boundary length among planning units not selected in the solution.

**Connectivity\_Fraction** The ratio of shared boundary length in the solution (`Connectivity_In`) to the total amount of boundary length (`Connectivity_Edge`). This ratio is an indicator of solution quality. Solutions with a lower ratio will have less planning units and will be more efficient.

### Value

`data.frame`

### See Also

`RapResults()`, `RapSolved()`.

**Examples**

```
# load data
data(sim_rs)

# show summary
summary(sim_rs)
```

---

update	<i>Update object</i>
--------	----------------------

---

**Description**

This function updates parameters or data stored in an existing `GurobiOpts()`, `RapUnreliableOpts()`, `RapReliableOpts()`, `RapData()`, `RapUnsolved()`, or `RapSolved()` object.

**Usage**

```
## S3 method for class 'GurobiOpts'
update(
  object,
  Threads = NULL,
  MIPGap = NULL,
  Method = NULL,
  Presolve = NULL,
  TimeLimit = NULL,
  NumberSolutions = NULL,
  MultipleSolutionsMethod = NULL,
  ...
)

## S3 method for class 'ManualOpts'
update(object, NumberSolutions = NULL, ...)

## S3 method for class 'RapData'
update(
  object,
  species = NULL,
  space = NULL,
  name = NULL,
  amount.target = NULL,
  space.target = NULL,
  pu = NULL,
  cost = NULL,
  status = NULL,
  ...
)
```

```

## S3 method for class 'RapReliableOpts'
update(object, BLM = NULL, failure.multiplier = NULL, max.r.level = NULL, ...)

## S3 method for class 'RapUnreliableOpts'
update(object, BLM = NULL, ...)

## S3 method for class 'RapUnsolOrSol'
update(object, ..., formulation = NULL, solve = TRUE)

```

## Arguments

object	<code>GurobiOpts()</code> , <code>RapUnreliableOpts()</code> , <code>RapReliableOpts()</code> , <code>RapData()</code> , <code>RapUnsolved()</code> , or <code>RapSolved()</code> object.
Threads	integer number of cores to use for processing.
MIPGap	numeric MIP gap specifying minimum solution quality.
Method	integer Algorithm to use for solving model.
Presolve	integer code for level of computation in presolve.
TimeLimit	integer number of seconds to allow for solving.
NumberSolutions	integer number of solutions to generate.
MultipleSolutionsMethod	integer name of method to obtain multiple solutions (used when <code>NumberSolutions</code> is greater than one). Available options are <code>"benders.cuts"</code> , <code>"solution.pool.0"</code> , <code>"solution.pool.1"</code> , and <code>"solution.pool.2"</code> . The <code>"benders.cuts"</code> method produces a set of distinct solutions that are all within the optimality gap. The <code>"solution.pool.0"</code> method returns all solutions identified whilst trying to find a solution that is within the specified optimality gap. The <code>"solution.pool.1"</code> method finds one solution within the optimality gap and a number of additional solutions that are of any level of quality (such that the total number of solutions is equal to <code>number_solutions</code> ). The <code>"solution.pool.2"</code> finds a specified number of solutions that are nearest to optimality. The search pool methods correspond to the parameters used by the Gurobi software suite (see <a href="https://www.gurobi.com/documentation/8.0/refman/poolsearchmode.html#parameter:PoolSearchMode">https://www.gurobi.com/documentation/8.0/refman/poolsearchmode.html#parameter:PoolSearchMode</a> ). Defaults to <code>"benders.cuts"</code> .
...	parameters passed to <code>update.RapReliableOpts()</code> , <code>update.RapUnreliableOpts()</code> , or <code>update.RapData()</code> .
species	integer or character denoting species for which targets or name should be updated.
space	integer denoting space for which targets should be updated.
name	character to rename species.
amount.target	numeric vector for new area targets (%) for the specified species.
space.target	numeric vector for new attribute space targets (%) for the specified species and attribute spaces.
pu	integer planning unit indices that need to be updated.

cost	numeric new costs for specified planning units.
status	integer new statuses for specified planning units.
BLM	numeric boundary length modifier.
failure.multiplier	numeric multiplier for failure planning unit.
max.r.level	numeric maximum R failure level for approximation.
formulation	character indicating new problem formulation to use. This can be either "unreliable" or "reliable". The default is NULL so that formulation in object is used.
solve	logical should the problem be solved? This argument is only valid for <a href="#">RapUnsolved()</a> and <a href="#">RapSolved()</a> objects. Defaults to TRUE.

**Value**

[GurobiOpts](#), [RapUnreliableOpts](#), [RapReliableOpts](#), [RapData](#), [RapUnsolved](#), or [RapSolved](#) object depending on argument to x.

**See Also**

[GurobiOpts](#), [RapUnreliableOpts](#), [RapReliableOpts](#), [RapData](#), [RapUnsolved](#), [RapSolved](#).

**Examples**

```
# load data
data(sim_ru, sim_rs)

# GurobiOpts
x <- GurobiOpts(MIPGap = 0.7)
y <- update(x, MIPGap = 0.1)
print(x)
print(y)

# RapUnreliableOpts
x <- RapUnreliableOpts(BLM = 10)
y <- update(x, BLM = 2)
print(x)
print(y)

# RapReliableOpts
x <- RapReliableOpts(failure.multiplier = 2)
y <- update(x, failure.multiplier = 4)
print(x)
print(y)

# RapData
x <- sim_ru@data
y <- update(x, space.target = c(0.4, 0.7, 0.1))
print(space.target(x))
print(space.target(y))

## RapUnsolved
```

```

x <- sim_ru
y <- update(x, amount.target = c(0.1, 0.2, 0.3), BLM = 3, solve = FALSE)
print(x@opts@BLM); print(amount.target(x))
print(y@opts@BLM); print(space.target(y))

## RapSolved
x <- sim_rs
y <- update(x, space.target = c(0.4, 0.6, 0.9), BLM = 100, Presolve = 1L,
            solve = FALSE)
print(x@opts@BLM); print(amount.target(x))
print(y@opts@BLM); print(space.target(y))

```

---

urap.proportion.held *Proportion held using unreliable RAP formulation.*

---

### Description

This is a convenience function to quickly calculate the proportion of variation that one set of points captures in a another set of points using the unreliable formulation.

### Usage

```
urap.proportion.held(x, y, y.weights = rep(1, nrow(y)))
```

### Arguments

x	<code>base::matrix()</code> of points
y	<code>base::matrix()</code> of points
y.weights	numeric vector of weights for each point in y. Defaults to equal weights for all points in y.

### Value

numeric value indicating the proportion of variation that x explains in y

### Examples

```
urap.proportion.held(as.matrix(iris[1:2,-5]), as.matrix(iris[1:5,-5]))
```

# Index

- \* **datasets**
  - casestudy\_data, 12
  - simulated\_data, 51
  
- adehabitatHR::mcp(), 19
- amount.held, 4
- amount.target, 5
- amount.target<-(amount.target), 5
- as.list, 6
- AttributeSpace, 7, 8, 9, 25
- AttributeSpace(), 8, 9, 26
- AttributeSpace-class, 7
- AttributeSpaces, 8
- AttributeSpaces(), 33
- AttributeSpaces-class, 9
  
- base::data.frame(), 11, 33–35, 37, 39
- base::environment(), 34, 35, 37, 39
- base::matrix(), 4, 13, 14, 19, 25, 31, 37, 39, 44, 46, 53, 66
- base::options(), 17
- basemap(), 26, 60
- blank.raster, 9
  
- calcBoundaryData, 10
- calcBoundaryData(), 21
- calcSpeciesAverageInPus, 11
- calcSpeciesAverageInPus(), 21
- casestudy\_data, 12
- cs\_pus (casestudy\_data), 12
- cs\_space (casestudy\_data), 12
- cs\_spp (casestudy\_data), 12
  
- DemandPoints, 7, 8, 13, 13
- DemandPoints(), 7, 14, 19
- DemandPoints-class, 14
- dp.subset, 14
  
- GurobiOpts, 15, 16, 41, 65
- GurobiOpts(), 6, 17, 29, 32, 33, 39, 40, 42, 47, 54, 63, 64
  
- GurobiOpts-class, 16
  
- hypervolume::hypervolume(), 19
  
- is.GurobiInstalled, 17
  
- ks::kde(), 19
  
- logging.file, 18
  
- make.DemandPoints, 19
- make.RapData, 20
- make.RapData(), 32, 34
- ManualOpts, 22, 22
- ManualOpts(), 22, 23, 39, 40
- ManualOpts-class, 23
- maximum.targets, 23
  
- names, 24
- names<- .RapData (names), 24
- names<- .RapUnsolOrSol (names), 24
  
- PBSmapping::PolySet(), 10, 28, 34, 35, 58
- PlanningUnitPoints, 7, 8, 25
- PlanningUnitPoints(), 7
- PlanningUnitPoints-class, 25
- plot, 26
- plot, RapSolved, missing-method (plot), 26
- plot, RapSolved, numeric-method (plot), 26
- plot, RapSolved, RapSolved-method (plot), 26
- PolySet (PolySet-class), 28
- PolySet-class, 28
- print, 28
- prob.subset, 29
- pu.subset, 30
  
- randomPoints, 31
- rap, 32
- RapData, 21, 33, 34, 40–43, 65

- RapData(), [5](#), [14](#), [15](#), [21](#), [24](#), [29–31](#), [33](#), [40](#), [42](#), [43](#), [47](#), [56](#), [57](#), [60](#), [61](#), [63](#), [64](#)
- RapData-class, [35](#)
- RapOpts (RapOpts-class), [35](#)
- RapOpts-class, [35](#)
- RapReliableOpts, [36](#), [36](#), [40–43](#), [65](#)
- RapReliableOpts(), [29](#), [32](#), [33](#), [35](#), [37](#), [40](#), [42](#), [43](#), [47](#), [63](#), [64](#)
- RapReliableOpts-class, [36](#)
- RapResults, [37](#), [38](#), [40](#), [41](#)
- RapResults(), [4](#), [5](#), [18](#), [29](#), [33](#), [37](#), [39](#), [40](#), [44–47](#), [54](#), [57](#), [62](#)
- RapResults-class, [38](#)
- RapSolved, [39](#), [40](#), [41](#), [65](#)
- RapSolved(), [4](#), [5](#), [14](#), [15](#), [18](#), [23](#), [24](#), [26](#), [27](#), [29–31](#), [33](#), [39](#), [40](#), [44–47](#), [52–54](#), [56](#), [57](#), [60–65](#)
- RapSolved-class, [40](#)
- raptr, [40](#)
- RapUnreliableOpts, [40](#), [41](#), [41](#), [42](#), [43](#), [65](#)
- RapUnreliableOpts(), [29](#), [32](#), [33](#), [35](#), [40–43](#), [47](#), [63](#), [64](#)
- RapUnreliableOpts-class, [42](#)
- RapUnsolved, [41](#), [42](#), [65](#)
- RapUnsolved(), [5](#), [14](#), [15](#), [23](#), [24](#), [29–31](#), [33](#), [39](#), [41](#), [42](#), [47](#), [52](#), [53](#), [56](#), [57](#), [60](#), [61](#), [63–65](#)
- RapUnsolved-class, [43](#)
- raster::brick(), [11](#)
- raster::raster(), [11](#), [12](#), [21](#), [31](#), [32](#), [49](#), [50](#)
- raster::stack(), [11](#), [12](#), [49](#), [51](#)
- RColorBrewer::brewer.pal(), [27](#), [60](#)
- read.RapResults(), [38](#), [39](#)
- rrap.proportion.held, [43](#)
- score, [44](#)
- selections, [45](#)
- show, [46](#)
- show, GurobiOpts-method (show), [46](#)
- show, ManualOpts-method (show), [46](#)
- show, RapData-method (show), [46](#)
- show, RapReliableOpts-method (show), [46](#)
- show, RapResults-method (show), [46](#)
- show, RapSolved-method (show), [46](#)
- show, RapUnreliableOpts-method (show), [46](#)
- show, RapUnsolved-method (show), [46](#)
- sim.pus, [48](#)
- sim.space, [49](#)
- sim.species, [50](#)
- sim\_rs (simulated\_data), [51](#)
- sim\_ru (simulated\_data), [51](#)
- simulated\_data, [51](#)
- solve, [52](#)
- solve, RapUnsolOrSol, GurobiOpts-method (solve), [52](#)
- solve, RapUnsolOrSol, logical-method (solve), [52](#)
- solve, RapUnsolOrSol, matrix-method (solve), [52](#)
- solve, RapUnsolOrSol, missing-method (solve), [52](#)
- solve, RapUnsolOrSol, numeric-method (solve), [52](#)
- SolverOpts (SolverOpts-class), [54](#)
- SolverOpts-class, [54](#)
- sp::Spatial, [9](#)
- sp::SpatialPoints(), [21](#), [34](#)
- sp::SpatialPointsDataFrame(), [21](#), [34](#)
- sp::SpatialPolygons(), [10](#), [11](#), [21](#), [32](#), [48–51](#), [58](#)
- sp::SpatialPolygonsDataFrame(), [10–12](#), [58](#)
- space.held, [54](#)
- space.plot, [55](#)
- space.target, [56](#)
- space.target<- (space.target), [56](#)
- SpatialPolygons2PolySet, [58](#)
- spp.plot, [59](#)
- spp.subset, [61](#)
- summary, [62](#)
- update, [63](#)
- update.RapData(), [64](#)
- update.RapReliableOpts(), [64](#)
- update.RapUnreliableOpts(), [64](#)
- urap.proportion.held, [66](#)