Package: robust.prioritizr (via r-universe)

May 29, 2026

Type Package

Title Robust Systematic Conservation Prioritization

Version 1.1.0

Description Systematic conservation prioritization with robust optimization techniques. This is important because conservation prioritizations typically only consider the most likely outcome associated with a conservation action (e.g., establishing a protected area will safeguard a threatened species population) and fail to consider other outcomes and their consequences for meeting conservation objectives. By extending the 'prioritizr' package, this package can be used to generate conservation prioritizations that account of uncertainty in the climate change scenario projections, species distribution models, ecosystem service models, and measurement errors. In particular, prioritizations can be generated to be fully robust to uncertainty by minimizing (or maximizing) objectives under the worst possible outcome. Since reducing the uncertainty associated with achieving conservation objectives may sacrifice other objectives (e.g., minimizing protected area implementation costs), prioritizations can also be generated to be partially robust based on a specified confidence level parameter. Partially robust prioritizations can be generated based on the chance constrained programming problem (Charnes & Cooper 1959, <doi:10.1287/mnsc.6.1.73>) and the conditional value-at-risk problem (Rockafellar & Uryasev 2000, <doi:10.21314/JOR.2000.038>).

Encoding UTF-8

License GPL (>= 3)

Language en-US

URL https://github.com/frankiecho/robust.prioritizr, https://frankiecho.github.io/robust.prioritizr/

BugReports https://github.com/frankiecho/robust.prioritizr/issues

VignetteBuilder knitr

LazyData true

Imports utils, parallel, Rcpp (>= 1.0.7), R6 (>= 2.5.1), rlang (>= 1.1.0), cli (>= 3.6.0), assertthat (>= 0.2.0), terra (>= 1.8.54), sf (>= 1.0-12), tibble (>= 2.0.0), units (>= 0.8.7), prioritizr (>= 8.1.0)

Suggests testthat (>= 3.1.0), knitr (>= 1.50), rmarkdown (>= 2.29), highs (>= 1.10.0-3)

Depends R (>= 4.1.0)

LinkingTo Rcpp (>= 1.0.7), RcppArmadillo (>= 0.10.7.3.0)

Collate 'RcppExports.R' 'internal.R' 'add_constant_robust_constraints.R' 'add_robust_min_set_objective.R' 'add_robust_min_shortfall_objective.R' 'add_variable_robust_constraints.R' 'data.R' 'get_feature_group_data.R' 'has_robust_constraints.R' 'import-standalone-all_finite.R' 'import-standalone-all_proportion.R' 'import-standalone-assertions_class.R' 'import-standalone-assertions_functions.R' 'import-standalone-assertions_handlers.R' 'import-standalone-assertions_misc.R' 'import-standalone-assertions_vector.R' 'package.R' 'robust_constraints.R' 'robust_objectives.R' 'run_example.R' 'transform_targets.R'

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Config/testthat/edition 3

Config/Needs/website tibble, stringr, ggplot2, tidyr, patchwork, tidyterra, dplyr, piggyback

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

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

Date/Publication 2026-05-29 03:51:04 UTC

RemoteUrl https://github.com/frankiecho/robust.prioritizr

RemoteRef HEAD

RemoteSha 7128bd662c2e74b78ccf359bfa8542e936d7d046

Contents

`add_constant_robust_constraints` *Add constant robust constraints*

Description

Add robust constraints to a conservation problem to specify that the solution should have the same minimum level of robustness for each feature group.

Usage

```
add_constant_robust_constraints(x, groups, conf_level = 1, target_trans = NA)
```

Arguments

`x`	`prioritizr::problem()` object.
`groups`	`character` vector indicating which features should be grouped together for the purposes of characterizing uncertainty. In particular, `groups` is used to specify a group name for each feature and features with the same group name will be grouped together. For example, if some of the features correspond to alternative predictions for the same species under different scenarios, then these features should have the same group name.
`conf_level`	`numeric` value describing the level of robustness required for the solution (ranging between 0 and 1). Defaults to 1, corresponding to a maximally robust solution. See the Details section for more information on this parameter.
`target_trans`	`character` value or vector of values specifying the method(s) for transforming and standardizing target thresholds for features that belong to the same feature group. Available options include computing the (`"mean"`) average, (`"min"`) minimum, or (`"max"`) maximum target threshold for each feature group. Additionally, (`"none"`) can be specified to ensure that the target thresholds considered during optimization are based on exactly the same values as specified when building the problem—even though different features in the same group may have different targets. Defaults to `NA` such that the average value is computed (similar to `target_trans = "mean"`) and a message indicating this behavior is displayed. If `target_trans` is a vector, then it must specify a transformation method for each feature group. For example, `c('min', 'max')` could be used to to specify that the target for the first feature group is calculated based on minimum value and the target for the second feature group is calculated based on the maximum value.

Details

The robust constraints are used to generate solutions that are robust to uncertainty. In particular, `conf_level` controls how important it is for a solution to be robust to uncertainty. To help explain how these constraints operate, we will consider the minimum set formulation of the reserve selection problem (per `prioritizr::add_min_set_objective()`. If `conf_level = 1`, then the solution must be maximally robust to uncertainty and this means that the solution must meet all of the targets for the features associated with each feature group. Although such a solution would be highly robust to uncertainty, it may not be especially useful because this it might have especially high costs (in other words, setting a high `conf_level` may result in a solution with a poor objective value). By lowering `conf_level`, this means that the solution must only meet certain percentage of the targets associated with each feature group. For example, if `conf_level = 0.95`, then the solution must meet, at least, 95% of the targets for the features associated with each feature group. Alternatively, if `conf_level = 0.5`, then the solution must meet, at least, half of the targets for the features associated with each feature group. Finally, if `conf_level = 0`, then the solution does not need to meet any of the targets for the features associated with each feature group. As such, it is not recommended to use `conf_level = 0`.

Value

An updated `prioritizr::problem()` object with the constraint added to it.

Data requirements

The robust constraints require that you have multiple alternative realizations for each biodiversity element of interest (e.g., species, ecosystems, ecosystem services). For example, we might have 5 species of interest. By applying different spatial modeling techniques, we might have 10 different models for each of the 5 different species We can use these models to generate 10 alternative realizations for each of the 5 species (yielding 50 alternative realizations in total). To use these data, we would input these 50 alternative realizations as 50 features when initializing a conservation planning problem (i.e., `prioritizr::problem()`) and then use this function to specify which of the of the features correspond to the same species (based on the feature groupings parameter).

See Also

See robust_constraints for an overview of all functions for adding robust constraints.

Other functions for adding robust constraints: `add_variable_robust_constraints()`

Examples

```
# Load packages
library(prioritizr)
library(terra)

# Get planning unit data
pu <- get_sim_pu_raster()

# Get feature data
features <- get_sim_features()
```

```
# Define the feature groups,
# Here, we will assign the first 2 features to the group A, and
# the remaining features to the group B
groups <- c(rep("A", 2), rep("B", nlyr(features) - 2))

# Build problem
p <-
  problem(pu, features) |>
  add_robust_min_set_objective() |>
  add_constant_robust_constraints(groups = groups, conf_level = 0.9) |>
  add_relative_targets(0.1) |>
  add_binary_decisions() |>
  add_default_solver(verbose = FALSE)

# Solve the problem
soln <- solve(p)

# Plot the solution
plot(soln)
```

`add_robust_min_set_objective` *Add robust minimum set objective*

Description

Add an objective to a conservation planning problem that minimizes the cost of the solution while ensuring that the solution is robust to uncertainty for each feature group.

Usage

```
add_robust_min_set_objective(x, method = "chance")
```

Arguments

`x`	`prioritizr::problem()` object.
`method`	`character` value with the name of the probabilistic constraint formulation method. Available options include the (`"chance"`) chance constraint programming method (Charnes and Cooper 1959) or (`"cvar"`) or the conditional value-at-risk method (Rockafellar and Uryasev 2000), Defaults to `"chance"`. See the Details section for further information on these methods.

Details

The robust minimum set objective seeks to find the set of planning units at a minimum cost such that the solution meets the targets in a robust manner for each feature group. Two methods are provided for formulating the optimization problem as a mixed integer linear programming problem. These

methods are the chance constraint programming method (`method = "chance"`) and conditional value-at-risk method (`method = "cvar"`). In particular, the chance constraint programming method is associated a more intuitive interpretation for the confidence level parameter (i.e., specified per `conf_level` with `add_constant_robust_constraints()` or `add_variable_robust_constraints()`). Whereas, the conditional value-at-risk constraint method may yield faster solve times. This is because the conditional value-at-risk constraint method preserves the convexity of an optimization problem, and uses continuous (instead of binary) auxiliary variables. Also note that the conditional value-at-risk constraint method may produce an infeasible solution for problems that are feasible with the chance constraint with the same `conf_level`. In such cases the chance chance constraint programming method should be used instead. As such, the chance constraint programming method may be more useful for facilitating stakeholder involvement for small-scale planning exercises, and the conditional value-at-risk constraint method may be more useful for large-scale applications.

Value

An updated `prioritizr::problem()` object with the objective added to it.

Mathematical formulation

This objective can be expressed mathematically for a set of planning units (I indexed by i), a set of feature groups (J indexed by j), and a set of features associated with each feature group (K indexed by k). Let c_i denote the cost of planning unit i, R_{ijk} the amount of feature k associated with planning unit i for feature group j, T_{jk} the target for each feature k in each feature group j, and α the confidence level for uncertainty (specified per `conf_level` with `add_constant_robust_constraints()` or `add_variable_robust_constraints()`). Additionally, to describe the decision variables, let x_i denote the status of the planning unit i (e.g., specifying whether planning unit i has been selected or not with binary values). Given this terminology, the robust minimum set formulation of the reserve selection problem is formulated as follows.

$$Minimize \sum_{i=1}^{I} x_i c_i subject to \Pr_k\{\sum_{i=1}^{I} x_i R_{ijk} \geq T_j\} \geq \alpha \quad \forall j \in J$$

Here, the objective function (first equation) is to minimize the total cost of the solution. The probabilistic constraints (second equation) specify that the solution must achieve a particular probability threshold (based on α) for meeting the targets of the features associated with each feature group. For example, if $\alpha = 1$, then each and every target associated with each feature group must be met. Alternatively, if $\alpha = 0.5$, then the solution must have a 50% chance of meeting the targets associated with each feature group. Approximation methods are used to linearize them so that the optimization problem can be solved with mixed integer programming exact algorithm solvers.

The chance constraint programming method uses a "big-M" formulation to linearize the probabilistic constraints (Charnes and Cooper 1959). To describe this method, let M_{jk} denote a binary auxiliary variable for each feature k associated with each feature group j. Also, let K_j denote a precomputed value describing the number of features associated with each feature group j. Given this terminology, the method involves replacing the probabilistic constraints with the following linear constraints.

$$\sum_{i=1}^{I}(x_i \times R_{ijk}) + (T_{jk} \times M_{jk}) \geq T_{jk} \quad \forall j \in J, k \in K \sum_{k=1}^{K_j} \frac{M_{jk}}{K_j} \leq 1 - \alpha \quad \forall j \in J M_{jk} \in \{0,1\} \quad \forall j \in J, k \in K$$

Here, the solution is allowed to fail to meet the targets for the features, and the auxiliary variable M_{jk} is used to calculate the proportion of features that do not have their targets met for each feature group. For a given feature group, the proportion of features that do not have their target met is constrained to be less than $1-\alpha$. This method allows for an intuitive interpretation of the confidence level parameter. Yet this method also adds $J \times K$ binary variables to the problem and, as such, may present long solve times for problems with many other decision variables and constraints.

The conditional value-at-risk constraint method presents a tighter formulation than the chance constraint programming method (Rockafellar and Uryasev 2000). As such, this method is able to better approximate the probabilistic constraints and, in turn, could potentially yield solutions that are more robust to uncertainty and less cost-efficient than the chance constraint programming method. To describe this method, let η_j denote a continuous auxiliary variable for each feature group j, and S_{jk} a continuous auxiliary variable for each feature k associated with each feature group j. Given this terminology, the method involves replacing the probabilistic constraints with the following linear constraints.

$$\sum_{i=1}^{I}(x_i \times R_{ijk}) - \eta_j + S_{jk} \geq 0 \quad \forall j \in J, k \in K \eta_j - \frac{1}{(1-\alpha) \times K_j} \sum_{k=1}^{K_j} S_{jk} \geq T_{jk} \quad \forall j \in J S_{jk} \geq 0 \quad \forall j \in J, k \in K \eta_j \in \mathbb{R}$$

Here, the continuous auxiliary variables are used to represent the "tail" of the distribution of the uncertain quantity (i.e., $\sum_{i=1}^{I} x_i r_{ijk}$). In other words, it ensures that the average of amount of each feature held by the solution for a particular feature group that falls below a particular quantile (i.e., $(1-\alpha)$) is greater than the target the feature (i.e., T_{jk}). Although this method does not provide an easily intuitive interpretation of the confidence level parameter, it only adds $J \times K + J$ continuous variables to the problem.

References

Charnes A & Cooper WW (1959) Chance-constrained programming. *Management Science*, 6(1), 73–79.

Rockafellar RT & Uryasev S (2000) Optimization of conditional value-at-risk. *Journal of Risk*, 2(3), 21–42.

See Also

See robust_objectives for an overview of all functions for adding robust objectives.

Other functions for adding robust objectives: `add_robust_min_shortfall_objective()`

Examples

```
# Load packages
library(prioritizr)
library(terra)

# Get planning unit data
pu <- get_sim_pu_raster()

# Get feature data
```

```
features <- get_sim_features()

# Define the feature groups,
# Here, we will assign the first 2 features to the group A, and
# the remaining features to the group B
groups <- c(rep("A", 2), rep("B", nlyr(features) - 2))

# Build problem with chance constraint programming method
p <-
  problem(pu, features) |>
  add_robust_min_set_objective(method = "cvar") |>
  add_constant_robust_constraints(groups = groups, conf_level = 0.9) |>
  add_binary_decisions() |>
  add_relative_targets(0.1) |>
  add_default_solver(verbose = FALSE)

# Solve the problem
soln <- solve(p)

# Plot the solution
plot(soln)
```

`add_robust_min_shortfall_objective`
Add robust minimum shortfall objective

Description

Add an objective to a conservation planning problem that minimizes the representation shortfalls for each feature group using the worst case shortfall or a quantile of the shortfall distribution, whilst ensuring that the total cost of the solution does not exceed a budget.

Usage

```
add_robust_min_shortfall_objective(x, budget)
```

Arguments

x	prioritizr::problem() object.
budget	numeric value specifying the maximum expenditure of the prioritization. For problems with multiple zones, the argument to budget can be (i) a single numeric value to specify a single budget for the entire solution or (ii) a numeric vector to specify a separate budget for each management zone.

Details

The robust minimum shortfall objective seeks to find the set of planning units that minimizes the representation shortfall for each feature group, subject to a budget. In particular, a target shortfall reflects difference between the target for a feature and the amount held by a candidate solution, expressed as a proportion of the target. These target shortfalls are then calculated for each of the features associated with a feature group, and a representation shortfall is used describe how well all the features associated with a particular feature group are represented by a candidate solution. Thus this objective aims to get as close as possible to reducing the representation shortfalls shortfalls to zero, by getting as close as possible to reaching all of the targets for the features associated with each of the feature groups. In the robust minimum shortfall formulation, the algorithm attempts to minimize the maximum shortfall within the feature group. In particular, the chance constraint programming method (Charnes and Cooper 1959) is used to formulate the optimization problem as a mixed integer linear programming problem. With this method, the confidence level parameter (i.e., specified per `conf_level` with `add_constant_robust_constraints()` or `add_variable_robust_constraints()`) describes the quantile of the target shortfalls associated with the feature group that should be minimized during optimization. For example, if `conf_level = 1` for a feature group, then the 100th quantile is used and this means that – after calculating the target shortfalls for each feature associated with the feature group – the largest target shortfall for the associated features is used to calculate the representation shortfall for the feature group. If `conf_level = 0.5` for a feature group, then the 50th quantile is used and this means that the median target shortfall for the features associated with the group is used to represent the representation shortfall for the feature group.

Value

An updated `prioritizr::problem()` object with the objective added to it.

Mathematical formulation

This objective can be expressed mathematically for a set of planning units (I indexed by i), a set of feature groups (J indexed by j), and a set of features associated with each feature group (K indexed by k). Let c_i denote the cost of planning unit i, R_{ijk} the amount of feature k associated with planning unit i for feature group j, T_{jk} the target for each feature k in each feature group j, W_{jk} the weight for each feature k in each feature group j, and α the confidence level for uncertainty (specified per `conf_level` with `add_constant_robust_constraints()` or `add_variable_robust_constraints()`). Additionally, to describe the decision variables, let x_i denote the status of the planning unit i (e.g., specifying whether planning unit i has been selected or not with binary values), v_{jk} the target shortfall for each feature k associated with each feature group j, and y_j the representation shortfall for for each feature group j. Given this terminology, the robust minimum shortfall formulation of the reserve selection problem is formulated as follows.

$$Minimize \sum_{j=1}^{J} y_j \times \frac{\sum_{k=1}^{K} W_{jk}}{K} subject to \sum_{i=1}^{I} x_i c_i \leq B \Pr_k \{ \sum_{i=1}^{I} (x_i \times R_{ijk}) + (T_{jk} \times v_{jk}) \geq T_{jk} \} \geq \alpha \quad \forall j \in J y_j \geq v_{jk} \quad \forall j$$

Here, the objective function (first equation) is to minimize the weighted sum of the representation shortfalls for each feature group. In particular, the representation shortfall for a given feature group is weighted according to the average weight value of the features associated with the feature group.

The budget constraints (second equation) ensure that the solution does not exceed the budget. The probabilistic constraints (third equation) specify that only some of the target shortfall variables (i.e., v_{jk}) associated with each feature group are used to calculate the representation shortfall for each feature group, and the subset of target shortfall variables that are used is based on the confidence level (i.e., α). For example, if $\alpha = 1$, then all of the target shortfall variables associated with each feature group must be used for the calculations. Alternatively, if $\alpha = 0.5$, then only enough of the target shortfall variables are required for the calculations to achieve a 50% chance of correctly calculating the target shortfall variables for a given feature group. The representation shortfall constraints (fourth equation) ensure that the representation shortfall variable for each feature group must be greater than or equal to the target shortfall variables of the features associated with the feature group. In combination with the other constraints, this means that the representation shortfall variable for a given feature group is calculated as the largest value of a subset of the target shortfall variables for the features associated with the feature group, and this particular subset is based on the confidence level. Thus if α is closer to a value of 1, then the representation shortfall variable for each feature group is calculated with a greater degree of certainty and, in turn, the optimization process seeks a solution that is more robust to uncertainty. Since the probabilistic constraints are non-linear, an approximation method is used to linearize them so that the optimization problem can be solved with mixed integer programming exact algorithm solvers.

The chance constraint programming method is used to linearize the probabilistic constraints (Charnes and Cooper 1959). To describe this method, let M_{jk} denote a binary auxiliary variable for each feature k associated with feature group j. Also K_j denote a pre-computed value describing the number of features associated with each feature group j. Given this terminology, the method involves replacing the probabilistic constraints with the following linear constraints.

$$\sum_{i=1}^{I}(x_i \times R_{ijk}) + (T_{jk} \times y_j) + (T_{jk} \times M_{jk}) \geq T_{jk} \quad \forall j \in J, k \in K \sum_{k=1}^{K_j} \frac{M_{jk}}{K_j} \leq 1 - \alpha \quad \forall j \in J M_{jk} \in \{0, 1\}$$

Here, the solution calculates the representation shortfall variable for a given feature group based on a particular subset of the target shortfalls for the associated features. Specifically, this subset based on a particular number of the smallest target shortfall variables based on α. For example, if a feature group is associated with 40 features and $\alpha = 0.75$, then the representation shortfall for the feature group is calculated by identifying which 10 of these 40 features have the largest target shortfall variables, and the shortfall variable used in the optimization process is the minimum of these large shortfall values. As such, the chance constraint programming method provides an intuitive approximation of the probabilistic constraints.

References

Charnes A & Cooper WW (1959) Chance-constrained programming. *Management Science*, 6(1), 73–79.

See Also

Other functions for adding robust objectives: `add_robust_min_set_objective()`

Examples

```
# Load packages
```

```
library(prioritizr)
library(terra)

# Get planning unit data
pu <- get_sim_pu_raster()

# Get feature data
features <- get_sim_features()

# Define the feature groups,
# Here, we will assign the first 2 features to the group A, and
# the remaining features to the group B
groups <- c(rep("A", 2), rep("B", nlyr(features) - 2))

# Build problem with budget calculated as 30% total cost
p <-
  problem(pu, features) %>%
  add_robust_min_shortfall_objective(
    budget = terra::global(pu, "sum", na.rm = TRUE)[[1]] * 0.3
  ) %>%
  add_constant_robust_constraints(groups = groups, conf_level = 0.4) %>%
  add_binary_decisions() %>%
  add_relative_targets(0.3) %>%
  add_default_solver(verbose = FALSE)

# Solve the problem
soln <- solve(p)

# Plot the solution
plot(soln)
```

`add_variable_robust_constraints` *Add variable robust constraints*

Description

Add robust constraints to a conservation problem to allow the solution's level of robustness to uncertainty to differ across feature groups. For example, this function may be especially useful when it is important to ensure that a prioritization is highly robust to uncertainty in the spatial distribution of threatened species, and only moderately robust to uncertainty in the spatial distribution of widespread species.

Usage

```
add_variable_robust_constraints(x, data)
```

Arguments

x	prioritizr::problem() object.
data	tibble::tibble() data frame containing information on the feature groups and confidence level associated with each group. Defaults to 1, corresponding to a maximally robust solution.

Details

The robust constraints are used to generate solutions that are robust to uncertainty. In particular, conf_level controls how important it is for a solution to be robust to uncertainty. To help explain how these constraints operate, we will consider the minimum set formulation of the reserve selection problem (per prioritizr::add_min_set_objective(). If conf_level = 1, then the solution must be maximally robust to uncertainty and this means that the solution must meet all of the targets for the features associated with each feature group. Although such a solution would be highly robust to uncertainty, it may not be especially useful because this it might have especially high costs (in other words, setting a high conf_level may result in a solution with a poor objective value). By lowering conf_level, this means that the solution must only meet certain percentage of the targets associated with each feature group. For example, if conf_level = 0.95, then the solution must meet, at least, 95% of the targets for the features associated with each feature group. Alternatively, if conf_level = 0.5, then the solution must meet, at least, half of the targets for the features associated with each feature group. Finally, if conf_level = 0, then the solution does not need to meet any of the targets for the features associated with each feature group. As such, it is not recommended to use conf_level = 0.

Value

An updated prioritizr::problem() object with the constraint added to it.

Data format

The data argument must be a tibble::tibble() data frame that has information on the feature groups and their confidence levels. Here, each row corresponds to a different feature group and columns contain information about the groups. In particular, data must have the following columns.

features A list column with the names of the features that belong to each group. In particular, if a particular set of features should belong to the same group, then they should be stored in the same element of this column.

conf_level A numeric column with values that describe the confidence level associated with each feature group (ranging between 0 and 1). See the Details section for information on conf_level values.

target_trans A character column with values that specify the method for transforming and standardizing target thresholds for each feature group. Available options include computing the ("mean") average, ("min") minimum, or ("max") maximum target threshold for each feature group. Additionally, ("none") can be specified to ensure that the target thresholds considered during optimization are based on exactly the same values as specified when building the problem—even though different features in the same group may have different targets. This column is option and if not provided then the target values will be transformed based on the average value for each feature group (similar to "mean") and a message indicating this behavior is displayed.

Data requirements

The robust constraints require that you have multiple alternative realizations for each biodiversity element of interest (e.g., species, ecosystems, ecosystem services). For example, we might have 5 species of interest. By applying different spatial modeling techniques, we might have 10 different models for each of the 5 different species We can use these models to generate 10 alternative realizations for each of the 5 species (yielding 50 alternative realizations in total). To use these data, we would input these 50 alternative realizations as 50 features when initializing a conservation planning problem (i.e., `prioritizr::problem()`) and then use this function to specify which of the of the features correspond to the same species (based on the feature groupings parameter).

See Also

Other functions for adding robust constraints: `add_constant_robust_constraints()`

Examples

```
# Load packages
library(prioritizr)
library(terra)
library(tibble)

# Get planning unit data
pu <- get_sim_pu_raster()

# Get feature data
features <- get_sim_features()

# Define the feature group data
# Here, we will assign the first 2 features to the group A, and the
# remaining features to the group B
groups <- c(rep("A", 2), rep("B", nlyr(features) - 2))

# Next, we will use this information to create a data frame containing
# the feature groups and specifying a confidence level of 0.95 for group A,
# and a confidence level of 0.5 for group B
constraint_data <- tibble(
  features = split(names(features), groups),
  conf_level = c(0.95, 0.5)
)

# Display constraint data
print(constraint_data)

# Build problem
p <-
  problem(pu, features) |>
  add_robust_min_set_objective() |>
  add_variable_robust_constraints(data = constraint_data) |>
  add_relative_targets(0.1) |>
  add_binary_decisions() |>
  add_default_solver(verbose = FALSE)
```

```
# Solve the problem
soln <- solve(p)

# Plot the solution
plot(soln)
```

`data` *Conservation planning dataset for Victoria, Australia*

Description

This is a conservation planning dataset for Victoria, Australia that can be used to generate robust prioritizations. This dataset was derived from Archibald *et al.* (2024), Department of Climate Change, Energy, the Environment and Water (2024), Global Administrative Areas (2024), and Williams *et al.* (2020). For an example of using this dataset, please refer to the the *Example using Victoria, Australia* vignette (`vignette("vic-cons-planning", package = "robust.prioritizr")`).

Usage

```
get_vic_study_area()

get_vic_cost()

get_vic_pa()

get_vic_species()

get_vic_species_metadata()
```

Format

`get_vic_study_area()` `sf::st_sf()` object.

`get_vic_cost()` `terra::rast()` object

`get_vic_species()` `terra::rast()` object

`get_vic_pa()` `terra::rast()` object

`vic_species_metadata()` `tibble::tibble()` object

Details

Briefly, this dataset contains 1.2988 × 104 planning units and 18 terrestrial vertebrate species For each species, the dataset contains the present-day spatial distribution for the species as well as projections for the species' future spatial distribution at 4 time periods based on 4 combinations of climate models and scenarios. To account for existing conservation efforts, the dataset also contains the locations of existing protected areas.

The following functions are provided to import the dataset:

`get_vic_study_area()` A `sf::st_sf()` object containing the spatial boundary of the Victoria, Australia (derived from Global Administrative Areas 2024).

`get_vic_cost()` A `terra::rast()` object containing the opportunity costs associated with protected area establishment (derived from Williams *et al.* 2020). This object contains a single layer, and grid cells denote planning units. Cells contain positive continuous values, such that greater values denote greater opportunity costs.

`get_vic_pa()` A `terra::rast()` object containing the locations of existing protected areas (derived from DCCEEW 2024). This object contains a single layer, and grid cells denote planning units. Cells contain binary values indicating if existing protected areas cover, at least, 50% of the grid cell or not.

`get_vic_species()` A `terra::rast()` object containing the present-day and potential future spatial distributions of terrestrial vertebrate species (derived from Archibald *et al.* 2024). This object contains a 306 layers, where each layer corresponds to the predicted spatial distribution of a particular species at a particular point in time based on a particular climate scenario. For a given layer, grid cells denote planning units. Cells contain binary values indicating if the species is predicted to be present or absent within the cell.

`get_vic_species_metadata()` A `tibble::tibble()` data frame containing information on the species' spatial distribution data (i.e., `get_vic_species()`). Here, each row of the `dadta.frame` corresponds to each layer of species' spatial distribution data, and columns describe different aspects of the layers. This object contains columns with the following values.

id `integer` index values for the layers. E.g., the fifth layer is associated with an `id` value of 5.

id `character` names for the layers. These values correspond to `names(get_vic_species())`.

species `character` scientific names of the species associated with the layers.

class `character` taxonomic classes of the species associated with the layers.

proj `character` names of the climate projections and timesteps associated with the layers. Layers that represent the species' present-day distributions are denoted with a value of `"historic_baseline_1990"`. Also, layers that represent the species' potential future distributions denoted with values of `"GCM-Ensembles_ssp126_2030"`, `"GCM- Ensembles_ssp126_2050"`, `"GCM-Ensembles_ssp126_2070"`, `"GCM- Ensembles_ssp126_2090"`, `"GCM-Ensembles_ssp245_2030"`, `"GCM- Ensembles_ssp245_2050"`, `"GCM-Ensembles_ssp245_2070"`, `"GCM-Ensembles_ssp245_2090"`, `"GCM-Ensembles_ssp370_2030"`, `"GCM-Ensembles_ssp370_2050"`, `"GCM-Ensembles_ssp370_2070"`, `"GCM-Ensembles_ssp370_2090"`, `"GCM-Ensembles_ssp585_2030"`, `"GCM-Ensembles_ssp585_2050"`, `"GCM-Ensembles_ssp585_2070"`, and `"GCM-Ensembles_ssp585_2090"`. Note that these values provide the same information as the `timestep` and `scenario` columns, and are provided to help with subsetting the data.

timestep `numeric` year of the datasets used to generate the layers. Layers that represent the species' present-day distributions are denoted with a year of `1990`, and layers that represent the species' potential future distributions denoted with years of `2030`, `2050`, `2070`, and `2090`.

scenario `character` names of the climate scenarios used to generate the layers. Layers that represent the species' present day distributions are denoted with `"historic_baseline"`. Also, layers that represent species' future distributions are associated with a particular Shared Socioeconomic Pathway (SSP) and Representative Concentration Pathways (RCP), such as (`"ssp126"`) SSP 1 and RCP 2.6, (`"ssp245"`) SSP 2 and RCP 4.5, (`"ssp370"`) SSP 3 and RCP 7.0, and (`"ssp585"`) SSP 5 and RCP 8.5.

sum `numeric` number of planning units where the species are predicted to be present within each of the layers.

References

Archibald CL, Summers DM, Graham EM, Bryan B (2024) Habitat suitability maps for Australian flora and fauna under CMIP6 climate scenarios. *GigaScience*, 13:giae002.

DCCEEW (2024), Collaborative Australian Protected Areas Database (CAPAD).

Global Administrative Areas (2024). Database of Global Administrative Areas. Version 4.1. Available at https://gadm.org (accessed on 15 August 2025).

Williams BA, Venter O, Allan JR, Atkinson SC, Rehbein JA, Ward M, Di Marco M, Grantham HS, Ervin J, Goetz SJ, Hansen AJ, Jantz P, Pillay R, Rodríguez-Buriticá S, Supples C, Virnig ALS, Watson JEM (2020) Change in terrestrial human footprint drives continued loss of intact ecosystems. *One Earth*, 3:371–382.

See Also

The code used to prepare this dataset are available online (https://github.com/jeffreyhanson/robust.prioritizr.data),

Examples

```
# load spatial R packages
library(sf)
library(terra)

# load data
vic_study_area <- get_vic_study_area()
vic_cost <- get_vic_cost()
vic_species <- get_vic_species()
vic_pa <- get_vic_pa()
 vic_species_metadata <- get_vic_species_metadata()

# preview data
print(vic_species_metadata)
print(vic_study_area)
print(vic_cost)
print(vic_species)
print(vic_pa)

# visualize data
plot(vic_study_area, main = "vic_study_area")
plot(vic_cost, main = "vic_cost")
plot(vic_species, main = "vic_species")
plot(vic_pa, main = "vic_pa")
```

robust.prioritizr *robust.prioritizr: Robust Systematic Conservation Prioritization in R*

Description

The **robust.prioritizr** R package provides tools for building and solving robust systematic conservation prioritization problems. It extends the **prioritizr** package to account for uncertainty in the input data. This is particularly useful when working with data that is subject to change, such as species distribution models under climate change scenarios.

Details

This package contains several vignettes that are designed to showcase its functionality. To view them, please use the code `vignette("name", package = "robust.prioritizr")` where `"name"` is the name of the desired vignette (e.g., `"robust.prioritizr"`).

robust.prioritizr Brief introduction to the package.

climate-sdm Example using simulated data from a species distribution model.

vic-cons-planning Example using Victoria, Australia.

Author(s)

Authors:

- Frankie Cho <frankie.cho@qut.edu.au> (ORCID)
- Jeffrey O Hanson <jeffrey.hanson@uqconnect.edu.au> (ORCID)

See Also

Useful links:

- Package website (https://frankiecho.github.io/robust.prioritizr/)
- Source code repository (https://github.com/frankiecho/robust.prioritizr)
- Report bugs (https://github.com/frankiecho/robust.prioritizr/issues)

robust_constraints *Add robust constraints*

Description

Add constraints to a conservation planning problem to explicitly account for uncertainty.

Details

The robust constraints functions are designed to be used with a robust objective function (e.g., add_robust_min_set_objective(), add_robust_min_shortfall_objective()). In particular, these functions are used to specify which features should be grouped together for characterizing plausible realizations. For example, if considering considering multiple projections for a species' distribution, these constraints are used to specify which features correspond to the same species. In addition to specifying feature groups, these constraints are also used to specify a confidence level that describes the level of robustness required for solutions.

The following robust constraint functions can be added to a conservation planning problem:

add_constant_robust_constraints() Add robust constraints to a conservation problem to specify that the solution should ideally aim for the same level of robustness for each feature group.

add_variable_robust_constraints() Add robust constraints to a conservation problem to specify that the solution should ideally aim for different levels of robustness for each feature group.

See Also

Other overviews: robust_objectives

Examples

```
# Load packages
library(prioritizr)
library(terra)
library(tibble)

# Get planning unit data
pu <- get_sim_pu_raster()

# Get feature data
features <- get_sim_features()

# Define the feature group data
# Here, we will assign the first 2 features to the group A, and the
# remaining features to the group B
groups <- c(rep("A", 2), rep("B", nlyr(features) - 2))

# Build problem with constant robust constraints
p1 <-
  problem(pu, features) |>
```

```
  add_constant_robust_constraints(groups = groups, conf_level = 0.5) |>
  add_robust_min_set_objective() |>
  add_relative_targets(0.1) |>
  add_binary_decisions() |>
  add_default_solver(verbose = FALSE)

# Next, we will create the input data for adding variable robust
# constraints. In particular, we specify a confidence level of 0.95 for
# group A, and a confidence level of 0.5 for group B
constraint_data <- tibble(
  features = split(names(features), groups),
  conf_level = c(0.95, 0.5)
)

# Display constraint data
print(constraint_data)

# Build problem with variable robust constraints
p2 <-
  problem(pu, features) |>
  add_variable_robust_constraints(data = constraint_data) |>
  add_robust_min_set_objective() |>
  add_relative_targets(0.1) |>
  add_binary_decisions() |>
  add_default_solver(verbose = FALSE)

# Solve the problems
soln <- c(solve(p1), solve(p2))
names(soln) <- c(
  "constant robust constraints", "variable robust constraints"
)

# Plot the solutions
plot(soln)
```

`robust_objectives` *Add a robust objective function*

Description

Add an objective function to a conservation planning problem that accounts for uncertainty.

Details

Robust objective functions are used to find solutions that are likely to meet conservation targets across a range of different scenarios or realizations of the input data. This is particularly useful when working with data that is uncertain, such as species distribution models under climate change scenarios. Note that robust constraints must also be used when using these objective functions (e.g., `add_constant_robust_constraints()`, `add_variable_robust_constraints()`).

The following robust objective functions can be added to a conservation planning problem:

`add_robust_min_set_objective()` Add an objective to a conservation planning problem to minimize the cost of the solution while ensuring that the targets for each feature group are met in a manner that is robust to uncertainty. This function provides a robust alternative to `prioritizr::add_min_set_objective()`.

`add_robust_min_shortfall_objective()` Add an objective to a conservation planning problem that minimizes the target shortfalls for each feature group in a manner that is robust to uncertainty, whilst ensuring that the total cost of the solution does not exceed a budget. This function provides a robust alternative to `prioritizr::add_min_shortfall_objective()`.

See Also

Other overviews: `robust_constraints`

Examples

```
# Load packages
library(prioritizr)
library(terra)

# Get planning unit data
pu <- get_sim_pu_raster()
features <- get_sim_features()

# Define the feature groups,
# Here, we will assign the first 2 features to the group A, and
# the remaining features to the group B
groups <- c(rep("A", 2), rep("B", nlyr(features) - 2))

# Build problem with robust min set objective
p1 <-
  problem(pu, features) %>%
  add_robust_min_set_objective() %>%
  add_constant_robust_constraints(groups = groups, conf_level = 0.4) %>%
  add_binary_decisions() %>%
  add_relative_targets(0.3) %>%
  add_default_solver(verbose = FALSE)

# Build problem with robust min shortfall objective,
# and budget set to 30% of the total cost of all planning units
p2 <-
  problem(pu, features) %>%
  add_robust_min_shortfall_objective(
    budget = terra::global(pu, "sum", na.rm = TRUE)[[1]] * 0.3
  ) %>%
  add_constant_robust_constraints(groups = groups, conf_level = 0.4) %>%
  add_binary_decisions() %>%
  add_relative_targets(0.3) %>%
  add_default_solver(verbose = FALSE)

# Solve problems
```

```
soln <- c(solve(p1), solve(p2))
names(soln) <- c("robust min set", "robust min shortfall")
plot(soln, axes = FALSE)
```

`run_example` *Run example?*

Description

Determine if the session is suitable for executing long-running examples.

Usage

```
run_example()
```

Details

This function will return `TRUE` if the session is interactive. Otherwise, it will only return `TRUE` if the session does not have system environmental variables that indicate that the session is being used for package checks, or for building documentation.

Value

A `logical` value.

Examples

```
# should examples be run in current environment?
run_example()
```

Index