Package: bifrost (via r-universe)

June 4, 2026

Title Branch-Level Inference Framework for Recognizing Optimal Shifts in Traits

Version 0.1.4

Description Methods for detecting and visualizing cladogenic shifts in multivariate trait data on phylogenies. Implements penalized-likelihood multivariate generalized least squares models, enabling analyses of high-dimensional trait datasets and large trees via searchOptimalConfiguration(). Includes a greedy step-wise shift-search algorithm following approaches developed in Smith et al. (2023) <doi:10.1111/nph.19099> and Berv et al. (2024) <doi:10.1126/sciadv.adp0114>. Methods build on multivariate GLS approaches described in Clavel et al. (2019) <doi:10.1093/sysbio/syy045> and implemented in the mvgls() function from the 'mvMORPH' package. Documentation and vignettes are available at <https://jakeberv.com/bifrost/>, including worked examples for the jaw-shape dataset.

License GPL (>= 2)

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

URL https://jakeberv.com/bifrost/, https://github.com/jakeberv/bifrost

BugReports https://github.com/jakeberv/bifrost/issues

Depends R (>= 4.1)

Imports ape, future, future.apply, progressr, phytools, grDevices, stats, mvMORPH, viridis, txtplot

Suggests RColorBrewer, boot, classInt, evd, fitdistrplus, geomorph, knitr, palaeoverse, parallel, patchwork, pbmcapply, phylolm, plotly, png, readxl, rmarkdown, scales, scatterplot3d, testthat (>= 3.0.0), univariateML, spelling, htmltools

Config/testthat/edition 3

Config/testthat/parallel false

Language en-US
Config/pak/sysreqs libglpk-dev libxml2-dev
Repository https://jakeberv.r-universe.dev
Date/Publication 2026-06-04 21:19:09 UTC
RemoteUrl https://github.com/jakeberv/bifrost
RemoteRef HEAD
RemoteSha 2c3477350657d8a3a4ab50d2bd9ceace4a82ebe2

Contents

`generateViridisColorScale` *Generate Scaled Viridis Color Palette for Rate Parameters*

Description

Creates a named color mapping for a set of numeric parameters (e.g., evolutionary rates) using the **viridis** color palette. Parameters are first sorted in ascending order and normalized to the range [0, 1], then mapped to evenly spaced viridis colors for intuitive visualization.

Usage

```
generateViridisColorScale(params)
```

Arguments

`params`	A named numeric vector of parameter values (e.g., rates). The names will be preserved and used to label the resulting color mapping.

Details

This function is useful for plotting results where parameters should be visually distinguished based on their magnitude (e.g., rate shifts across a phylogeny). By using the perceptually uniform viridis palette, it avoids misleading color interpretations common with rainbow scales.

Value

A named list with two elements:

`NamedColors` A named character vector of hex color codes, with names corresponding to the input parameter names, ordered by increasing parameter value.

`ParamColorMapping` A named numeric vector of the sorted parameter values, maintaining the same order and names as `NamedColors`.

See Also

`viridis::viridis()` for details on the color palette.

Examples

```
if (requireNamespace("viridis", quietly = TRUE)) {
  library(viridis)
  set.seed(1)
  rates <- c(A = 0.1, B = 0.5, C = 0.9)
  color_scale <- generateViridisColorScale(rates)

  # View the color assignments
  color_scale$NamedColors

  # Plot with colors
  barplot(color_scale$ParamColorMapping,
          col = color_scale$NamedColors,
          main = "Rates with Viridis Colors")
}
```

---

plot.rateMap *Plot* rateMap *Objects*

Description

Render a computed `"rateMap"` object. The usual workflow is `x <- rateMap(...)` followed by `plot(x, ...)`.

Usage

```
## S3 method for class 'rateMap'
plot(
  x,
  value = "value",
  palette = NULL,
  reverse_palette = NULL,
  color_mode = NULL,
  n_categories = NULL,
```

```
  category_bin_method = NULL,
  category_breaks = NULL,
  category_labels = NULL,
  ncolors = NULL,
  legend_title = NULL,
  legend = NULL,
  fsize = NULL,
  tip_fsize = NULL,
  legend_fsize = NULL,
  ftype = NULL,
  show_tip_labels = TRUE,
  outline = FALSE,
  lwd = 3,
  type = c("phylogram", "fan", "arc"),
  mar = rep(0.3, 4),
  direction = "rightwards",
  offset = NULL,
  xlim = NULL,
  ylim = NULL,
  hold = TRUE,
  underscore = FALSE,
  arc_height = 2,
  legend_digits = NULL,
  ...
)
```

Arguments

x	An object of class "rateMap" returned by rateMap() or rateMapView().
value	Character column in the plotted summary table (x$intervals) to map to branch colors. The default "value" plots the central estimate chosen by rateMap(). When uncertainty = TRUE, useful alternatives include "mean", "median", "sd", "ci_width", "highest_density_interval_width", and "cv".
palette	Optional palette override used for this plot. This can be an hcl.colors() palette name, a vector of colors, or a palette function.
reverse_palette	Optional logical override for palette reversal used for this plot.
color_mode	Optional color-mode override. Use "continuous" for a numeric color ramp or "category" for ordered discrete rate categories. If NULL, the stored mode in x is used. Category mode draws one color per exact value or category bin; continuous mode draws a many-color ramp.
n_categories	Optional category-count override for color_mode = "category". This changes the target number of category bins, not the number of colors in continuous mode.
category_bin_method	Optional category-binning override for color_mode = "category". Use "pretty" for pretty breaks or "equal" for equal-width numeric intervals. Ignored when category_breaks is supplied.

`category_breaks`
: Optional category-break override for `color_mode = "category"`. Custom breaks override automatic bins.

`category_labels`
: Optional category-label override for `color_mode = "category"`.

`ncolors`
: Optional number of colors to use when recoloring this plot with `color_mode = "continuous"`. If omitted, the stored `x$ncolors` value is used, falling back to `256` for older objects.

`legend_title`
: Optional legend title override for this plot.

`legend`
: Legend length. If `NULL` or `TRUE`, a layout-specific default is used. Set `legend = FALSE` to suppress the legend. Continuous legends are drawn with `phytools::add.color.bar()`; category legends are drawn by `rateMap` so bin labels can reflect rate categories and diagnostics.

`fsize`
: Numeric font-size vector. The first element is passed as the tip label `fsize` to `phytools::plotSimmap()` and, when `outline = TRUE`, to `phytools::plotTree()`. The second element is used by the legend.

`tip_fsize`
: Optional override for the first `fsize` element passed to the underlying tree plot.

`legend_fsize`
: Optional override for the legend font size.

`ftype`
: Font type. The first element is passed as `ftype` to `phytools::plotSimmap()` and, when `outline = TRUE`, to `phytools::plotTree()`. The second element is reserved for legends.

`show_tip_labels`
: Logical; if `FALSE`, `rateMap` sets the tree-plot `ftype` to `"off"` before calling the underlying `phytools` plotter.

`outline`
: Logical; if `TRUE`, draw a branch outline beneath the rate map. The outline pass is drawn with `phytools::plotTree()` before the colored `phytools::plotSimmap()` pass.

`lwd`
: Branch and legend line widths. The first element is passed to `phytools::plotSimmap()` and, with `+ 2`, to `phytools::plotTree()` for outlines. The second element is used for the legend.

`type`
: Plot type: `"phylogram"`, `"fan"`, or `"arc"`. This is passed to `phytools::plotSimmap()` for fan and arc layouts and to `phytools::plotTree()` for outline passes.

`mar`
: Plot margins passed to `phytools::plotSimmap()` and, when `outline = TRUE`, to `phytools::plotTree()`.

`direction`
: Plotting direction for `type = "phylogram"`; passed to `phytools::plotSimmap()` and `phytools::plotTree()`.

`offset`
: Tip-label offset passed to `phytools::plotSimmap()` and, when `outline = TRUE`, to `phytools::plotTree()`.

`xlim, ylim`
: Optional plot limits passed to `phytools::plotSimmap()` and, when `outline = TRUE`, to `phytools::plotTree()`.

`hold`
: Logical controlling `rateMap`'s device hold/flush guard. `rateMap` passes `hold = FALSE` to the internal `phytools` calls so the two-pass outline/color drawing is controlled in one place.

underscore	Logical; if FALSE, underscores in tip labels may be shown as spaces by the underlying phytools::plotSimmap() and phytools::plotTree() calls.
arc_height	Arc height passed through to phytools::plotSimmap() and, for outlines, phytools::plotTree() when type = "arc".
legend_digits	Optional number of digits for legend endpoint labels. If omitted, small-magnitude values use enough digits to avoid zero-valued legend endpoints.
...	Additional arguments are rejected. Include all display choices as named plot() arguments.

Details

The plot method uses phytools::plotSimmap() and draws either a continuous color-bar legend or a segmented rate-category color bar. The plotting controls intentionally mirror the phytools density-map plotting style for phylogram, fan, and arc layouts. In branch-summary category mode, near-zero or high-outlier rate diagnostics are drawn as special categories when rate-valued columns are plotted; these special categories do not consume positions in the ordered palette used for regular rate bins. When special categories are present, the category legend spans the full plotted value range and marks the diagnostic cutoff separating special and regular bins. When plotting non-rate columns such as "sd", diagnostic columns are preserved only as metadata with rate_flag_source provenance; special rate categories are not drawn for those non-rate values. The selected value column must contain finite values for every plotted interval; uncertainty columns such as "sd" may be all NA for single-fit objects and are rejected with a clear error.

The plot() method handles "phylogram", "fan", and "arc" layouts. legend controls the length of the color bar; set legend = FALSE to suppress it. legend_digits controls numeric endpoint labels and defaults to enough precision to avoid rounding small values to zero. Single fitted objects should be converted explicitly with rateMap() before plotting, for example plot(rateMap(search_a), ...).

Relationship to phytools plotting arguments. plot.rateMap() keeps the tree-layout argument names close to phytools: type, fsize, ftype, lwd, mar, direction, offset, xlim, ylim, underscore, and arc_height are forwarded to phytools::plotSimmap() or phytools::plotTree() as described above. rateMap fixes phytools::plotSimmap() options such as colors, pts, node.numbers, add, and hold internally, because those are determined by the computed "rateMap" object and by the optional outline pass. palette, color_mode, n_categories, category_breaks, category_labels, legend_title, legend_digits, and the rate-flag display behavior are rateMap controls, not phytools arguments.

Value

Invisibly returns the plotted "rateMap" object.

References

Revell, L. J. (2013). Two new graphical methods for mapping trait evolution on phylogenies. *Methods in Ecology and Evolution*, 4, 754-759.

Revell, L. J. (2024). phytools 2.0: an updated R ecosystem for phylogenetic comparative methods (and other things). *PeerJ*, 12, e16505. doi:10.7717/peerj.16505

See Also

rateMap(), phytools::densityMap(), phytools::plotSimmap(), phytools::plotTree()

Examples

```
## Not run:
rm_obj <- rateMap(fits, progress = FALSE)
plot(
  rm_obj,
  type = "arc",
  show_tip_labels = FALSE,
  legend_fsize = 0.8
)

# If rm_obj was built with uncertainty = TRUE, plot uncertainty directly:
plot(rm_obj, value = "sd", palette = "Inferno")

# Use a continuous ramp instead of the default ordered rate categories:
plot(rm_obj, color_mode = "continuous")

# Or keep category colors but change the binning:
plot(rm_obj, n_categories = 5, category_bin_method = "equal")
plot(rm_obj, category_breaks = c(-4, -2, 0, 2))

## End(Not run)
```

---

plot_ic_acceptance_matrix

Plot IC Acceptance Matrix with Optional Rate-of-Improvement Overlay

---

Description

Create a two-layer base R plot that visualizes information criterion (IC) scores across a sequence of sub-model evaluations, highlighting which steps were *accepted* vs *rejected*. Optionally, a secondary y-axis overlays the **rate of improvement** (first difference of IC scores) as a line with markers.

Usage

```
plot_ic_acceptance_matrix(
  matrix_data,
  plot_title = "IC Acceptance Matrix Scatter Plot",
  plot_rate_of_improvement = TRUE,
  rate_limits = c(-400, 150),
  baseline_ic = NULL
)
```

Arguments

matrix_data	A two-column matrix or data.frame. Column 1 must be numeric IC scores in evaluation order; Column 2 must be a logical or numeric flag (0/1) indicating whether the step was accepted.
plot_title	character(1). Title to draw above the plot.
plot_rate_of_improvement	logical(1). If TRUE, overlay the first differences of the IC series on a secondary (right) y-axis along with a horizontal reference line at zero.
rate_limits	numeric(2). Y-axis limits for the rate-of-improvement overlay (i.e., diff(IC)), used only when plot_rate_of_improvement = TRUE. Defaults to c(-400, 150).
baseline_ic	Optional numeric(1). If provided, this value is used as the baseline IC score (step 1) in place of matrix_data[1, 1] for plotting and for computing diff(IC). Default is NULL (use matrix_data[1, 1]).

Details

The function expects a two-column object where:

- Column 1 contains the IC score at each step (numeric; lower is better).
- Column 2 contains an indicator for acceptance (0 = rejected, 1 = accepted).

The first IC value is treated as the *baseline* and is plotted as a larger black point with a numeric label. If baseline_ic is supplied, it is used as the baseline IC score (step 1) in place of matrix_data[1, 1] for both the baseline annotation and the rate-of-improvement series (diff(IC)). This is useful because matrix_data begins with the first evaluated shift model (rather than the true no-shift baseline). To achieve this behavior, pass the true baseline via baseline_ic to avoid labeling the first evaluated model as the baseline.

Accepted steps are drawn as blue filled points connected by a thin line; rejected steps are drawn as small red crosses. When plot_rate_of_improvement = TRUE, the function overlays a secondary y-axis on the right that shows diff(IC) values (the per-step change in IC; more negative implies improvement).

The function uses only base graphics. It sets plot margins and mgp via par(), and (when overlaying) uses par(new = TRUE) to layer the IC plot over the rate-of-improvement axes. Initial user par is reset on exit.

Axes and scaling. Tick marks for the primary (IC) x/y axes are computed with pretty() to give clean bounds. The secondary axis for the rate of improvement uses rate_limits (default c(-400, 150)); adjust via the argument if your expected diff(IC) range differs substantially.

Value

Invisibly returns NULL. Called for its plotting side effects.

See Also

par, plot, axis, lines, points, legend, mtext, title

Examples

```
ic <- c(-1000, -1012, -1008, -1025, -1020, -1030)
accepted <- c(1, 0, 1, 0, 1)  # steps 2..6 relative to baseline
mat <- cbind(ic, c(1, accepted))  # mark baseline as accepted for plotting
plot_ic_acceptance_matrix(mat, plot_title = "IC Path")
# Avoid non-ASCII glyphs in titles on CRAN/CI:
plot_ic_acceptance_matrix(mat, plot_rate_of_improvement = TRUE)
# Override baseline IC:
plot_ic_acceptance_matrix(mat, baseline_ic = -995)
```

---

print.bifrost_search *Print method for bifrost search results*

Description

Prints a compact summary of a completed Bifrost search, including the baseline and optimal information criterion (IC) values, the inferred shift node set, key search settings, and (when present) optional diagnostics such as IC-history and IC-weight support.

Usage

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

Arguments

x	A bifrost_search object returned by searchOptimalConfiguration().
...	Unused (S3 compatibility).

Value

Invisibly returns x. Called for its printing side effects.

---

print.rateMap *Print a* rateMap *Object*

Description

Print a concise summary of a "rateMap" object, including the number of fits, summary mode, target/check mode, weighting mode, uncertainty status, color mode, plotted value, value range, rate-flag diagnostics when active, and uncertainty ranges when available.

Usage

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

Arguments

x	An object of class "rateMap" returned by rateMap().
...	Ignored.

Value

Invisibly returns x.

---

rateMap *Compute Branchwise Rate Maps Across Runs*

Description

Summarize fitted regime-specific rates across a list of stochastic-map-aware model fits or completed bifrost_search results. By default, rateMap() follows bifrost's branch-level framing: the first retained tree is used as the plotting scaffold, all retained trees must match in topology and branch lengths, each branch receives one summarized log-rate, and runs are averaged with equal weight.

Usage

```
rateMap(
  fits,
  weights = c("equal", "ic"),
  uncertainty = FALSE,
  summary = c("branch", "interval"),
  log = TRUE,
  value_summary = c("mean", "median"),
  target_tree = NULL,
  workers = NULL,
  progress = TRUE,
  control = rateMapControl()
)
```

Arguments

fits	A completed run, fitted model object, or non-empty list of these objects. Supported shapes are bifrost_search objects, mvgls objects, list(model = <mvgls>), and scratch-style lists with variables$tree plus param. Single supported objects are wrapped automatically as a convenience for one-fit plotting and inspection.
weights	Fit-level weighting mode. "equal" gives every retained fit equal weight. "ic" computes standard IC weights from optimal_ic and requires all retained fits to have the same IC_used. A numeric vector is also accepted and treated as custom fit weights.
uncertainty	Logical; if TRUE, compute and return across-fit uncertainty summaries for every summary row: whole branches when summary = "branch" and depth-grid intervals when summary = "interval".

summary	Character; "branch" computes one length-weighted value per edge, while "interval" slices branches on a global depth grid.
log	Logical; if TRUE (the default), transform extracted rate parameters with log() before branch, interval, and across-fit averaging. Weighted means are then mean log-rates, equivalent to the log of weighted geometric mean rates. Set log = FALSE to summarize rates in their original units.
value_summary	Character; central estimate stored in the summary table column intervals$value. "mean" uses the weighted mean. "median" uses the weighted median. When log = TRUE, both summaries are computed on the log-rate scale.
target_tree	Optional explicit target tree used as the summary and plotting scaffold. This may be any target or summary tree with the same topology and tip labels as the inputs. It does not need to contain stochastic maps because rateMap() replaces maps with display maps in the returned object. Branch lengths in target_tree define the geometry of the returned and plotted tree.
workers	Optional number of future workers to use. If NULL, the current future::plan() is used as-is.
progress	Logical; if TRUE, display a text progress bar via progressr::with_progress().
control	A "rateMap_control" object from rateMapControl(), or a named list of advanced control options.

Details

Everyday workflow. rateMap() is a compute function: call it to build a reusable "rateMap" object, then call plot(x, ...) to choose what to display. Common compute controls are fits, weights, uncertainty, summary, log, and value_summary. Common display controls belong in plot() or rateMapView(), including value, type, palette, color_mode, n_categories, show_tip_labels, and legend sizing. Controls such as target, check, res, tree_fun, param_fun, and the future_* arguments live in rateMapControl() because they are mainly advanced tools for same-topology tree samples, interval summaries, custom fit objects, and larger run sets.

Algorithmic provenance. rateMap() is inspired by phytools::densityMap(), which summarizes a set of stochastic maps by slicing mapped branches on a shared depth grid and coloring a SIMMAP-style tree by an averaged branchwise quantity. Here the averaged quantity is not a posterior probability of a mapped state; instead, each mapped state is translated through the corresponding fitted regime-rate parameter from each run. The plotting interface likewise follows the phytools density-map family by returning a colored SIMMAP tree and drawing it with phytools::plotSimmap() plus either a segmented rate-category color bar or a continuous color-bar legend.

Although rateMap() is designed for summarizing multiple fitted maps, it also accepts a single completed bifrost search or supported fit. Single-fit input is a convenience for inspecting one fitted model before scaling up to multi-run summaries.

rateMap() can also summarize same-topology posterior or sensitivity samples where branch lengths differ. In that case, usually supply an explicit target_tree and use control = list(check = "topology"). The target can be any tree with the same topology and tip labels as the inputs; an MCC tree is only one possible choice. The target tree supplies the plotted topology and branch lengths, while rates are matched from each input tree by descendant-tip clade keys rather than by edge order.

Summary modes. With the default `summary = "branch"` and `log = TRUE`, each edge receives one length-weighted average log-rate from each run before the across-run summary is computed. This is the natural default for `bifrost` searches because shifts are placed at nodes, so a `bifrost` branch is not expected to change regimes internally. With `summary = "interval"`, each target-tree branch is subdivided by the global depth grid controlled by `control = list(res = ...)`. If source branch lengths differ from the target branch length, source stochastic-map segments are projected onto target intervals by relative position along the matched branch. Interval mode is useful for general stochastic maps that can genuinely change state along a branch.

Log-rate averaging. When `log = TRUE`, rate parameters are transformed before branch-level, interval-level, and across-fit summaries are computed. With weights `w_i`, the default plotted mean is therefore `sum(w_i * log(rate_i))`, which is the log of a weighted geometric mean. It is not `log(sum(w_i * rate_i))`, the log of a weighted arithmetic mean. Use `log = FALSE` when downstream interpretation requires arithmetic summaries on the original rate scale. Raw fitted rate parameters must be strictly positive in either mode. Negative plotted values are therefore valid in log-rate maps whenever the positive raw fitted rates are less than one.

Tree checks and targets. In `rateMapControl()`, `check = TRUE` is equivalent to `check = "full"` and requires topology and branch lengths to match the target tree. Use `check = "topology"` when all inputs have the same topology and tip labels but may have different branch lengths. `check = FALSE` or `check = "none"` skips the upfront `ape::all.equal.phylo()` check, but every target branch must still be recoverable in every input tree by descendant-tip set. This function is not a mixed-topology posterior summarizer; clades absent from an input tree are treated as an error rather than being marginalized over topology.

When `target_tree` is supplied, it is used directly as the plotting scaffold and need not contain SIMMAP maps. When `target_tree = NULL`, `rateMapControl(target = "first")` uses the first retained input tree, and `rateMapControl(target = "mcc")` chooses the retained input tree with the highest sum of log clade credibilities. For truly same-topology inputs, the MCC score is usually tied, so `"mcc"` commonly resolves to the first retained tree. MCC target selection does not make `rateMap()` a mixed-topology summarizer: every target branch must still be present in every retained input. If you already have a preferred consensus, chronogram, MCC, maximum-likelihood, or otherwise curated target tree, pass it with `target_tree`.

Fit weights. `weights = "equal"` assigns the same weight to each retained fit. `weights = "ic"` computes standard information-criterion weights from each retained fit's `optimal_ic`, requiring all retained fits to share the same non-missing `IC_used`. A numeric `weights` vector can be supplied for custom weighting. Weights are subset to retained fits after `rateMapControl(na_action = "omit")` and are normalized to sum to one. IC weights are descriptive fit-level weights for comparable retained searches; they do not choose a formal threshold or make incomparable searches comparable.

Uncertainty summaries. The returned `intervals` data frame is the plotted summary table. With `summary = "branch"`, it has one row per branch. With `summary = "interval"`, it has one row per plotted depth-grid interval. When `uncertainty = TRUE`, this table includes across-fit summaries for each plotted row: weighted mean, weighted median, weighted standard deviation, quantiles, highest-density interval bounds, quantile and highest-density interval widths, coefficient of variation, and the number of finite run-level values. The run-level values are also returned in `run_values` as one matrix per edge. These are weighted empirical summaries of run-level values, not posterior uncertainty or model-internal uncertainty unless the retained inputs themselves have that interpretation. For posterior tree samples, use `weights = "equal"` if each retained run represents one posterior draw. `value_summary` controls whether the plotted `value` column uses the weighted mean or

weighted median. These summaries are computed on the log-rate scale when `log = TRUE`. Highest-density intervals use the same shortest empirical interval calculation for both equal and unequal fit weights. For `weights = "equal"`, `sd` is the ordinary sample standard deviation of retained run-level values. For unequal weights, `sd` is the square root of the normalized weighted variance, `sum(w * (x - mu)^2)`, with weights normalized to sum to one.

Display views. Returned objects include a default category-style display mapping so they can be plotted immediately. Palette, category-bin, legend title, and alternative-value choices are intentionally controlled by `plot(x, ...)` or `rateMapView()`, not by `rateMap()`. For a single `bifrost` search with `log = FALSE`, category display is the closest formal analogue to the illustrative `generateViridisColorScale()` plot. For multi-run summaries, displayed categories are bins for summarized branch values; they should not be read as newly inferred `bifrost` regimes. When `summary = "branch"` and `color_mode = "category"`, the `rate_categories` table also reports branch-level summaries for the plotted values assigned to each bin. These summaries are recomputed by `rateMapView()` or `plot()` whenever the plotted value or category breaks change. They are not computed for `summary = "interval"` because interval rows depend on the plotting grid rather than on discrete biological branch units.

Rate diagnostics. Branch-summary maps compute rate diagnostics through `rateMapControl(rate_flags = rateMapRateFlags())`. The default `rateMapRateFlags()` setting records the fitted-rate range and fold range but applies no special near-zero or high-outlier rule. Supplying `zero_floor`, equivalently `rateMapRateFlags(method = "floor", zero_floor = ...)`, flags finite rates at or below an explicit manual floor. Use `rateMapRateFlags(method = "tail_cluster")` to apply a deterministic, Otsu-style guarded two-class split on sorted log rates when the near-zero values form a broader separated lower-tail cluster rather than one extreme adjacent gap. The split is kept only when gap, fold-reduction, tail-fraction, and minimum-count guardrails are satisfied. These flags never remove branches and never alter `intervals$value`; they add `rate_flag` metadata, `rate_flag_source` provenance, object-level `rate_diagnostics`, and, in category display mode, special display categories such as `"near-zero"`. Special categories are colored outside the ordered palette, so the remaining regular categories still use the full low-to-high palette range. In category legends, diagnostic cutoffs mark where special categories end and regular bins begin. Set `rate_flags = NULL` to turn off rate-flag metadata and diagnostics entirely.

Value

An object of class `"rateMap"` with components:

`tree` A SIMMAP-style tree whose mapped segments encode color-bin indices in continuous mode, or named rate categories in category mode.

`cols` The resolved color palette. In continuous mode this has length `ncolors`; in category mode it has one color per exact value or category bin.

`lims` Numeric length-2 vector giving the plotted value range.

`breaks` Numeric vector of palette bin boundaries in continuous mode, or category boundaries/values in category mode.

`values` List of plotted-row central values by edge before color binning.

`intervals` Plotted summary table. With `summary = "branch"`, this has one row per branch. With `summary = "interval"`, this has one row per plotted depth-grid interval. When branch-level rate diagnostics are enabled, this table also includes `rate_for_flagging`, `rate_flag`, `rate_flag_source`, `is_near_zero`, and `is_high_outlier`.

`rate_categories` Data frame describing discrete rate categories when `color_mode = "category"`; otherwise `NULL`. With `summary = "branch"`, this table also includes bin-level summaries of the plotted branch values, including `n_branches`, `value_mean`, `value_median`, `value_min`, `value_max`, `value_sd`, and `total_branch_length`.

`run_values` When `uncertainty = TRUE`, list of numeric matrices containing run-level values for each edge. Matrix rows match the plotted rows for that edge and columns are retained fits. Otherwise `NULL`.

`clade_key` Character descendant-tip key for each target-tree edge.

`edge_matches` Integer matrix mapping target-tree edge rows to matched source-tree edge rows for each retained fit.

`summary` The summary mode used, `"interval"` or `"branch"`.

`uncertainty` Logical indicating whether uncertainty summaries were computed.

`value_summary` Central estimate used for the plotted summary table column `intervals$value`.

`quantile_probs` Quantile probabilities used for uncertainty summaries.

`highest_density_interval_prob` Highest-density interval mass used for uncertainty summaries.

`rate_diagnostics` List summarizing rate-flag settings, counts, detected tail cutoffs, and fold-rate ranges with and without flagged branches.

`rate_flags` The normalized `"rateMap_rate_flags"` control object used for rate diagnostics.

`rate_flag_source` Character name of the rate-valued column used to compute or preserve `rate_flag` metadata, or `NA` when diagnostics are disabled.

`plot_value` Current interval column mapped to branch colors.

`target` Target-tree selection mode used.

`check` Tree compatibility check mode used.

`weights` Normalized fit weights used for aggregation.

`weight_mode` Weighting mode used: `"equal"`, `"ic"`, or `"custom"`.

`weight_table` Data frame linking retained input indices, weights, and IC values when available.

`palette` Original palette specification.

`reverse_palette` Logical indicating whether the palette was reversed.

`ncolors` Stored continuous-ramp resolution used when recoloring with `color_mode = "continuous"` and no explicit `ncolors`.

`color_mode` Coloring mode used for the current tree.

`n_categories` Category count target used when `color_mode = "category"`.

`category_breaks` Category breaks or exact category values used when `color_mode = "category"`.

`category_labels` Category labels used when `color_mode = "category"`.

`category_bin_method` Automatic category-binning method used when `color_mode = "category"` and `category_breaks = NULL`.

`title` Legend title used for plotting.

`n_fits` Number of fits used after validation or omission.

`omitted` Integer indices of omitted fits when `na_action = "omit"`.

References

Paradis, E., and Schliep, K. (2019). ape 5.0: an environment for modern phylogenetics and evolutionary analyses in R. *Bioinformatics*, 35, 526-528. doi:10.1093/bioinformatics/bty633

Clavel, J., Aristide, L., and Morlon, H. (2019). A penalized likelihood framework for high-dimensional phylogenetic comparative methods and an application to new-world monkeys brain evolution. *Systematic Biology*, 68, 93-116. doi:10.1093/sysbio/syy045

Revell, L. J. (2013). Two new graphical methods for mapping trait evolution on phylogenies. *Methods in Ecology and Evolution*, 4, 754-759.

Revell, L. J. (2024). phytools 2.0: an updated R ecosystem for phylogenetic comparative methods (and other things). *PeerJ*, 12, e16505. doi:10.7717/peerj.16505

See Also

`plot.rateMap()`, `rateMapView()`, `rateMapControl()`, `phytools::densityMap()`, `phytools::plotSimmap()`

Examples

```
## Not run:
# A list of completed bifrost searches can be summarized directly:
rm_obj <- rateMap(list(search_a, search_b, search_c))
plot(rm_obj, type = "arc", show_tip_labels = FALSE)

# A single completed bifrost search can be inspected with the same API:
one_search_map <- rateMap(search_a)
plot(one_search_map, type = "arc", show_tip_labels = FALSE)

# Scratch-style lists remain supported:
scratch_fit <- list(
  variables = list(tree = mapped_tree),
  param = c("0" = 0.12, "1" = 0.45)
)
rateMap(list(scratch_fit))

# Use bifrost model-level IC weights across comparable sensitivity runs:
rm_ic <- rateMap(list(search_a, search_b), weights = "ic")

# Branch-level, discrete-category maps are the bifrost default:
branch_rates <- rateMap(list(search_a, search_b))
branch_rates$rate_categories

# To mimic the old jaw-shape preview style for a single search, use raw
# fitted rates and category colors:
jaw_view <- rateMapView(rateMap(search_a, log = FALSE), palette = viridis::viridis)

# Category bins use pretty breaks by default. Use equal-width bins or custom
# boundaries when those are easier to compare across figures:
equal_bin_rates <- rateMapView(
  branch_rates,
  n_categories = 5,
  category_bin_method = "equal"
```

```
)
custom_bin_rates <- rateMapView(
  branch_rates,
  category_breaks = c(-4, -2, 0, 2),
  category_labels = c("slow", "middle", "fast")
)

# Continuous interval maps remain available for stochastic maps with
# along-branch changes:
interval_rates <- rateMap(
  list(search_a, search_b),
  summary = "interval",
  control = list(res = 200)
)
plot(interval_rates, color_mode = "continuous")

# Custom fit weights are normalized internally:
custom_weighted <- rateMap(
  list(search_a, search_b, search_c),
  weights = c(2, 1, 1),
  summary = "branch"
)

# Posterior trees with the same topology but different branch lengths can be
# summarized on any explicit target or summary tree:
posterior_target_rates <- rateMap(
  posterior_fit_list,
  target_tree = summary_tree,
  summary = "branch",
  weights = "equal",
  uncertainty = TRUE,
  control = list(check = "topology")
)
plot(posterior_target_rates, value = "sd", type = "arc")

# Or choose the retained input tree with the highest summed log clade
# credibility as the plotting scaffold:
posterior_mcc_rates <- rateMap(
  posterior_fit_list,
  summary = "branch",
  control = list(check = "topology", target = "mcc")
)

# Large sensitivity sets can be explicitly subsampled before plotting.
# By default, rates are mapped on the log scale:
set.seed(1)
idx <- sample(seq_along(fit_list), size = 1000)
rm_sub <- rateMap(
  fit_list[idx],
  workers = 8,
  control = list(res = 100, future_strategy = "multisession")
)
```

```
# Use log = FALSE only when a raw-rate scale is preferred:
rm_raw <- rateMap(fit_list[idx], log = FALSE)
plot(
  rm_sub,
  type = "arc",
  show_tip_labels = FALSE,
  lwd = 1,
  palette = c("lightblue", "blue", "pink", "red")
)

## End(Not run)
```

`rateMapControl` *Control Advanced* `rateMap()` *Options*

Description

Build a control object for less common `rateMap()` settings. These options are useful for interval maps, same-topology tree samples, custom fit object shapes, invalid-fit handling, and future-based parallel scheduling.

Usage

```
rateMapControl(
  res = 100,
  check = TRUE,
  target = c("first", "mcc"),
  tree_fun = NULL,
  param_fun = NULL,
  na_action = c("error", "omit"),
  rate_flags = rateMapRateFlags(),
  quantile_probs = c(0.025, 0.975),
  highest_density_interval_prob = 0.95,
  future_strategy = c("multisession", "multicore"),
  future_seed = FALSE,
  future_scheduling = 1,
  future_chunk_size = NULL
)
```

Arguments

`res`	Integer resolution of the global depth grid used to subdivide target-tree branches when `summary = "interval"`. Ignored by `summary = "branch"`.
`check`	Logical or character check mode. `TRUE` or `"full"` verifies that extracted trees and the target tree match in topology and branch lengths. `"topology"` verifies matching topology/tip labels while allowing branch lengths to differ. `FALSE` or `"none"` skips the upfront `ape::all.equal.phylo()` check, but target branches still must be matchable by descendant-tip sets.

target — Character target-tree selection when target_tree = NULL in rateMap(). "first" uses the first retained input tree. "mcc" chooses the retained input tree with the highest sum of log clade credibilities.

tree_fun — Optional function used to extract a mapped tree from each element of fits. If NULL, common bifrost and mvgls shapes are auto-detected. For "bifrost_search" objects, the default uses tree_no_uncertainty_untransformed to preserve the original branch-length scale; supply tree_fun explicitly to map a different tree field.

param_fun — Optional function used to extract a named numeric vector of state-specific fitted rates from each element of fits. If NULL, common bifrost and mvgls shapes are auto-detected.

na_action — What to do when a run has invalid parameters. "error" stops immediately. "omit" drops invalid runs before aggregation.

rate_flags — A "rateMap_rate_flags" object from rateMapRateFlags(), or a named list of rate-flagging options. Use NULL to disable rate diagnostics.

quantile_probs — Numeric length-2 vector of quantile probabilities to report when uncertainty = TRUE.

highest_density_interval_prob — Numeric scalar giving the highest-density interval mass to report when uncertainty = TRUE.

future_strategy — Future backend used only when workers is supplied to rateMap(). Must be one of "multisession" or "multicore".

future_seed — Seed control passed to future.apply::future_lapply().

future_scheduling — Scheduling control passed to future.apply::future_lapply().

future_chunk_size — Chunk size passed to future.apply::future_lapply().

Value

A "rateMap_control" object for the control argument of rateMap().

Examples

```
## Not run:
# Same-topology tree samples with differing branch lengths:
ctrl <- rateMapControl(check = "topology")
posterior_rates <- rateMap(
  posterior_fit_list,
  target_tree = summary_tree,
  uncertainty = TRUE,
  control = ctrl
)

# Interval maps with a finer depth grid:
interval_rates <- rateMap(
  fits,
```

```
  summary = "interval",
  control = rateMapControl(res = 200)
)

## End(Not run)
```

`rateMapRateFlags` *Control Near-Zero and Tail Rate Diagnostics for* `rateMap()`

Description

Build a rate-flagging control object for `rateMapControl()`. These options identify branch-rate summaries that are effectively zero, or optionally in a separated high-rate tail, without deleting or changing the fitted rates. Flags are reported in the returned `intervals` table and summarized in `rate_diagnostics`.

Usage

```
rateMapRateFlags(
  near_zero = NULL,
  high_outlier = FALSE,
  method = NULL,
  zero_floor = NULL,
  cluster_min_log_gap = log(10),
  cluster_min_fold_reduction = 10,
  cluster_max_tail_fraction = 0.4,
  cluster_min_flagged = 3,
  cluster_min_regular = 10,
  zero_label = "near-zero",
  high_label = "high-outlier",
  zero_color = "grey70",
  high_color = "black"
)
```

Arguments

`near_zero`	Logical or `NULL`. If `TRUE`, flag lower-tail or floor-level rates. The default `NULL` resolves to `FALSE` for `method = "none"` and `TRUE` for `method = "floor"` or `method = "tail_cluster"`.
`high_outlier`	Logical; if `TRUE`, flag isolated upper-tail rates.
`method`	Optional detection method. `NULL` chooses `"floor"` when `zero_floor` is supplied and `"none"` otherwise. `"floor"` uses `zero_floor` for near-zero rates. `"tail_cluster"` uses an Otsu-style guarded two-class split of log rates to identify a separated lower-tail cluster, and, when `high_outlier = TRUE`, a separated upper-tail cluster. `"none"` computes diagnostics without adding special rate flags. Inactive switches are normalized to `FALSE`: `method = "none"` sets

near_zero and high_outlier to FALSE, and method = "floor" sets high_outlier to FALSE.

zero_floor
: Optional non-negative rate floor. When supplied with method = NULL, the method resolves to "floor". Finite rates less than or equal to this value are flagged as near-zero.

cluster_min_log_gap
: Minimum log-rate gap required between a cluster-defined tail and the remaining regular rates.

cluster_min_fold_reduction
: Minimum fold-range reduction required after separating a cluster-defined tail from the regular rates.

cluster_max_tail_fraction
: Maximum fraction of positive finite rates that can be assigned to a cluster-defined tail.

cluster_min_flagged
: Minimum number of branches required for a cluster-defined tail.

cluster_min_regular
: Minimum number of branches that must remain in the regular rate set after separating a cluster-defined tail.

zero_label, high_label
: Labels used for flagged display categories.

zero_color, high_color
: Colors used for flagged display categories in category mode.

Details

The "tail_cluster" method is an Otsu-style diagnostic adapted to sorted branch log rates. It evaluates two-class splits and keeps a tail only when guardrails for log-rate gap size, fold-range reduction, tail fraction, and minimum counts are all satisfied. It is display metadata, not data deletion, model correction, or formal threshold selection.

Value

A normalized "rateMap_rate_flags" object for rateMapControl(rate_flags =).

References

Otsu, N. (1979). A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1), 62-66. doi:10.1109/TSMC.1979.4310076

Examples

```
## Not run:
ctrl <- rateMapControl(rate_flags = rateMapRateFlags(zero_floor = 1e-8))
rm_obj <- rateMap(fits, control = ctrl)
rm_obj$rate_diagnostics

cluster_ctrl <- rateMapControl(
```

```
  rate_flags = rateMapRateFlags(method = "tail_cluster")
)
rm_obj <- rateMap(fits, control = cluster_ctrl)
rm_obj$rate_diagnostics

## End(Not run)
```

`rateMapView` *Create a Display View of a* `rateMap` *Object*

Description

Recompute the display mapping for a computed `"rateMap"` object without drawing it. This is optional in the usual `rateMap()` then `plot()` workflow; use it when you want to save or inspect category tables, color bins, or an uncertainty-valued map object and then reuse the same display mapping in one or more plots. The numeric branch or interval summaries computed by `rateMap()` are not recomputed.

Usage

```
rateMapView(
  x,
  value = "value",
  palette = NULL,
  reverse_palette = NULL,
  color_mode = NULL,
  n_categories = NULL,
  category_bin_method = NULL,
  category_breaks = NULL,
  category_labels = NULL,
  ncolors = NULL,
  legend_title = NULL
)
```

Arguments

`x`	An object of class `"rateMap"` returned by `rateMap()`.
`value`	Name of the numeric column in `x$intervals` to map to colors. The default, `"value"`, uses the central summary computed by `rateMap()`.
`palette`	Optional palette override. This can be an `hcl.colors()` palette name, a vector of colors, or a palette function.
`reverse_palette`	Optional logical override for palette reversal.
`color_mode`	Optional color-mode override. Use `"category"` for discrete ordered bins or `"continuous"` for a continuous ramp.

`n_categories` Optional target category count for `color_mode = "category"`.

`category_bin_method`
Optional automatic category-binning method: `"pretty"` or `"equal"`.

`category_breaks`
Optional strictly increasing category boundaries.

`category_labels`
Optional labels for displayed categories.

`ncolors` Optional number of colors for continuous ramps. If omitted, the stored `x$ncolors` value is used, falling back to `256` for older objects.

`legend_title` Optional legend title stored on the returned object.

Value

A `"rateMap"` object with updated tree maps, color palette, `intervals$value`, `rate_categories`, and legend title. For branch-summary category views, `rate_categories` includes bin-level summaries of the plotted branch values and is recomputed whenever the view changes. For branch-summary maps with active rate diagnostics, diagnostic flags are recomputed for rate-valued views (`"value"`, `"mean"`, or `"median"`) and preserved as metadata for non-rate views such as `"sd"`. When preserved for a non-rate view, `rate_flag_source` identifies the rate-valued column that the flags classify; the flags do not classify the displayed uncertainty value. The selected `value` column must contain finite values for every plotted interval; uncertainty columns such as `"sd"` may be all `NA` for single-fit objects and are rejected with a clear error.

Examples

```
## Not run:
rm_obj <- rateMap(fits, uncertainty = TRUE)
display_obj <- rateMapView(
  rm_obj,
  palette = "Viridis",
  n_categories = 5,
  legend_title = "Mean log fitted rate"
)
plot(display_obj, type = "arc", show_tip_labels = FALSE)

sd_obj <- rateMapView(
  rm_obj,
  value = "sd",
  palette = "Inferno",
  color_mode = "continuous"
)

## End(Not run)
```

searchOptimalConfiguration *Search for an Optimal Multi-Regime (Shift) Configuration on a Phylogeny*

Description

Greedy, stepwise search for evolutionary regime shifts on a phylogeny using multivariate `mvgls` fits from **mvMORPH**. The routine:

1. builds one-shift candidate trees for all internal nodes meeting a tip-size threshold (via `generatePaintedTrees`),
2. fits each candidate in parallel and ranks them by improvement in the chosen information criterion (IC; `GIC` or `BIC`),
3. iteratively adds shifts that pass a user-defined acceptance threshold,
4. optionally revisits accepted shifts to prune overfitting using a small IC tolerance window,
5. optionally computes per-shift IC weights by refitting the model with each shift removed.

Models are fitted directly in multivariate trait space (no PCA), assuming a multi-rate Brownian Motion with proportional VCV scaling across regimes. Extra arguments in `...` are forwarded to `mvgls`. In practice, `method` and `error` are often the most important of these: the package vignettes use `method = "H&L"` for intercept-only, high-dimensional response matrices and `method = "LL"` for formula-based searches with predictors, while `error = TRUE` asks `mvgls()` to estimate a nuisance measurement-error (intraspecific-variance) term from the data.

Usage

```
searchOptimalConfiguration(
  baseline_tree,
  trait_data,
  formula = "trait_data ~ 1",
  min_descendant_tips,
  num_cores = 2,
  ic_uncertainty_threshold = 1,
  shift_acceptance_threshold = 1,
  uncertaintyweights = FALSE,
  uncertaintyweights_par = FALSE,
  plot = FALSE,
  IC = "GIC",
  store_model_fit_history = TRUE,
  verbose = FALSE,
  ...
)
```

Arguments

`baseline_tree` A rooted `phylo` (or SIMMAP/`phylo`) object representing the starting tree. It does not need to already be painted: the function coerces the input to a `phylo` object and internally paints a single baseline state at the root before generating candidate shift configurations. Tip labels must match `trait_data`.

`trait_data` A `matrix` or `data.frame` of continuous trait values with row names matching `baseline_tree$tip.label` (same order). For the default `formula = "trait_data ~ 1"`, `trait_data` is typically supplied as a numeric matrix, but a numeric response-only `data.frame` is also accepted. When using more general formulas (e.g., pGLS-style models), a `data.frame` with named columns can be used instead.

`formula` Character string or formula object passed to `mvgls`. Defaults to `"trait_data ~ 1"`, which fits an intercept-only model treating the supplied multivariate trait matrix as the response. This is the appropriate choice for most morphometric data where there are no predictor variables. For more general models, `formula` can reference subsets of `trait_data` explicitly, for example `"trait_data[, 1:5] ~ 1"` to treat columns 1-5 as a multivariate response, `"trait_data[, 1:5] ~ trait_data[, 6]"` to fit a multivariate pGLS with an indexed predictor, or `cbind(y1, y2) ~ size + grp` to fit a named-column pGLS with numeric or factor predictors.

`min_descendant_tips`
Integer ($\geq$1). Minimum number of tips required for an internal node to be considered as a candidate shift (forwarded to `generatePaintedTrees`). Larger values reduce the number of candidate shifts by excluding very small clades. For empirical datasets, values around `10` are a reasonable starting choice and can be tuned in sensitivity analyses.

`num_cores` Integer. Number of workers for candidate scoring. Uses plain serial evaluation when `num_cores = 1`. For `num_cores > 1`, uses `future::plan(multicore)` on Unix outside `RStudio`; otherwise uses `future::plan(multisession)`. During the parallel candidate-scoring blocks, BLAS/OpenMP threads are capped to 1 (per worker) to avoid CPU oversubscription.

`ic_uncertainty_threshold`
Numeric ($\geq$0). Reserved for future development in post-search pruning and uncertainty analysis; currently not used by `searchOptimalConfiguration()`.

`shift_acceptance_threshold`
Numeric ($\geq$0). Minimum IC improvement (baseline - new) required to accept a candidate shift during the forward search. Larger values yield more conservative models. For analyses based on the Generalized Information Criterion (`"GIC"`), a threshold on the order of `20` units is a conservative choice that tends to admit only strongly supported shifts. Simulation studies in Berv et al. (2026) suggest that this choice yields good balanced accuracy between detecting true shifts and avoiding false positives, but users should explore alternative thresholds in sensitivity analyses for their own datasets.

`uncertaintyweights`
Logical. If `TRUE`, compute per-shift IC weights serially by refitting the optimized model with each shift removed in turn. Exactly one of `uncertaintyweights`

or uncertaintyweights_par must be TRUE to trigger IC-weight calculations; setting both to TRUE will result in an error. When enabled, the per-shift weights are returned in the $ic_weights component of the result.

uncertaintyweights_par
: Logical. As above, but compute per-shift IC weights in parallel using **future.apply**. Exactly one of uncertaintyweights or uncertaintyweights_par must be TRUE to trigger IC-weight calculations.

plot
: Logical. If TRUE, draw/update a SIMMAP plot as the search proceeds (requires **phytools**).

IC
: Character. Which information criterion to use, one of "GIC" or "BIC" (case-sensitive).

store_model_fit_history
: Logical. If TRUE, store a per-iteration record of fitted models, acceptance decisions, and IC values. To keep memory usage low during the search, per-iteration results are written to a temporary directory (tempdir()) and read back into memory at the end of the run.

verbose
: Logical. If TRUE, report progress during candidate generation and model fitting. By default, progress is emitted via message(). When plot = TRUE in an interactive RStudio session, progress is written via cat() so it remains visible while plots are updating. Set to FALSE to run quietly (default). Use suppressMessages() (and capture.output() if needed) to silence or capture output.

...
: Additional arguments passed to mvgls (e.g., method, penalty, target, error, REML, etc.). In the workflows emphasized in the package vignettes, method = "H&L" is used for intercept-only searches on high-dimensional response matrices, whereas method = "LL" is used for formula-based searches with predictors and should also be used when IC = "BIC". In **mvMORPH**, method = "H&L" is restricted to intercept-only models and the "RidgeArch" penalty. Setting error = TRUE asks mvgls() to estimate a nuisance measurement-error (intraspecific-variance) term from the data.

Details

Input requirements.

- *Tree:* baseline_tree should be a rooted phylo tree with branch lengths interpreted in units of time. An ultrametric tree is not required. The starting tree does not need to already be painted; searchOptimalConfiguration() paints a single baseline regime internally before building shifted candidates.
- *Trait data alignment:* rownames(trait_data) must match baseline_tree$tip.label in both names and order; any tips without data should be pruned beforehand.
- *Data type:* trait_data is typically a numeric matrix of continuous traits; numeric response-only data.frames are also supported for intercept-only searches, and named mixed-type data.frames are supported for richer formulas. High-dimensional settings ($p \geq n$) are supported via penalized-likelihood mvgls() fits.

Search outline.

1. *Baseline:* Fit `mvgls` on the baseline tree (single regime) to obtain the baseline IC.
2. *Candidates:* Build one-shift trees for eligible internal nodes (`generatePaintedTrees`); fit each with `fitMvglsAndExtractGIC.formula` or `fitMvglsAndExtractBIC.formula` (internal helpers; not exported) and rank by ΔIC.
3. *Greedy add:* Add the top candidate, refit, and accept if $\Delta\text{IC} \geq$ `shift_acceptance_threshold`; continue down the ranked list.
4. *Optional IC weights:* If `uncertaintyweights` (or `uncertaintyweights_par`) is `TRUE`, compute an IC weight for each accepted shift by refitting the final model with that shift removed and comparing the two ICs via `aicw`.

Parallelization. When `num_cores = 1`, candidates are scored serially. For larger values, candidate sub-model fits are distributed with **future** + **future.apply**. On Unix outside `RStudio`, `multicore` is used; otherwise `multisession` is used. A sequential plan is restored afterward.

Plotting. If `plot = TRUE`, trees are rendered with `plotSimmap()`; shift IDs are labeled with `nodelabels()`.

Regime VCVs. The returned `$VCVs` are extracted from the fitted multi-regime model via `extractRegimeVCVs` and reflect regime-specific covariance estimates (when `mvgls` is fitted under a PL/ML method).

For high-dimensional trait datasets ($p \geq n$), penalized-likelihood settings in `mvgls()` are often required for stable estimation. The package vignettes distinguish two common workflows. For intercept-only searches on high-dimensional response matrices (for example, GPA-aligned landmark data), the jaw-shape vignette uses `method = "H&L"` with the default `"RidgeArch"` penalty; in **mvMORPH**, this is a fast approximation to penalized LOOCV and is only available for intercept-only models. For formula-based searches with predictors, the avian skeleton vignette uses `method = "LL"` instead. When `IC = "BIC"`, `method = "LL"` should be used. Across empirical workflows, `error = TRUE` is often a sensible default because it asks `mvgls()` to estimate a nuisance measurement-error (intraspecific-variance) term from the data. Users should consult the **mvMORPH** documentation for details on available methods and penalties and tune these choices to the structure of their data.

Value

A named `list` with (at minimum):

- `user_input`: captured call (as a list) for reproducibility.
- `tree_no_uncertainty_transformed`: SIMMAP tree from the optimal (no-uncertainty) model on the transformed scale used internally by `mvgls`.
- `tree_no_uncertainty_untransformed`: same topology with original edge lengths restored.
- `model_no_uncertainty`: the final `mvgls` model object.
- `shift_nodes_no_uncertainty`: integer vector of accepted shift nodes.
- `optimal_ic`: final IC value; `baseline_ic`: baseline IC.
- `IC_used`: `"GIC"` or `"BIC"`; `num_candidates`: count of candidate one-shift models evaluated.
- `model_fit_history`: if `store_model_fit_history = TRUE`, a list of per-iteration fits (loaded from temporary files written during the run) and an `ic_acceptance_matrix` (IC value and acceptance flag per step).
- `VCVs`: named list of regime-specific VCV matrices extracted from the final model (penalized-likelihood estimates if PL was used).

Additional components appear conditionally:

- `ic_weights`: a `data.frame` of per-shift IC weights and evidence ratios when `uncertaintyweights` or `uncertaintyweights_par` is `TRUE`.
- `warnings`: character vector of warnings/errors encountered during fitting (if any).

Convergence and robustness

The search is greedy and may converge to a local optimum. Use a stricter `shift_acceptance_threshold` to reduce overfitting, and re-run the search with different `min_descendant_tips` and IC choices (`"GIC"` vs `"BIC"`) to assess stability of the inferred shifts. For a given run, the optional IC-weight calculations (`uncertaintyweights` or `uncertaintyweights_par`) can be used to quantify support for individual shifts. It is often helpful to repeat the analysis under slightly different settings (e.g., thresholds or candidate-size constraints) and compare the resulting sets of inferred shifts.

Note

Internally, this routine coordinates multiple unexported helper functions: `generatePaintedTrees`, `fitMvglsAndExtractGIC.formula`, `fitMvglsAndExtractBIC.formula`, `addShiftToModel`, `removeShiftFromTree`, and `extractRegimeVCVs`. Through these, it may also invoke lower-level utilities such as `paintSubTree_mod` and `paintSubTree_removeShift`. These helpers are internal implementation details and are not part of the public API.

See Also

`mvgls`, `GIC`, `BIC`, `plot_ic_acceptance_matrix` for visualizing IC trajectories and shift acceptance decisions, and `generateViridisColorScale` for mapping regime-specific rates or parameters to a viridis color scale when plotting trees; packages: **mvMORPH**, **future**, **future.apply**, **phytools**, **ape**.

Examples

```
library(ape)
library(phytools)
library(mvMORPH)
set.seed(1)

# Simulate a tree
tr <- pbtree(n = 50, scale = 1)

# Define two regimes: "0" (baseline) and "1" (high-rate) on a subset of tips
states <- setNames(rep("0", Ntip(tr)), tr$tip.label)
high_clade_tips <- tr$tip.label[1:20]
states[high_clade_tips] <- "1"

# Make a SIMMAP tree for the BMM simulation
simmap <- phytools::make.simmap(tr, states, model = "ER", nsim = 1)

# Simulate traits under a BMM model with ~10x higher rate in regime "1"
sigma <- list(
  "0" = diag(0.1, 2),
```

```
  "1" = diag(1.0, 2)
)
theta <- c(0, 0)

sim <- mvMORPH::mvSIM(
  tree  = simmap,
  nsim  = 1,
  model = "BMM",
  param = list(
    ntraits = 2,
    sigma   = sigma,
    theta   = theta
  )
)

# mvSIM returns either a matrix or a list of matrices depending on mvMORPH version
X <- if (is.list(sim)) sim[[1]] else sim
rownames(X) <- simmap$tip.label

# Run the search on the unpainted tree (single baseline regime)
res <- searchOptimalConfiguration(
  baseline_tree              = as.phylo(simmap),
  trait_data                 = X,
  formula                    = "trait_data ~ 1",
  min_descendant_tips        = 10,
  num_cores                  = 1,   # keep it simple / CRAN-safe
  shift_acceptance_threshold = 20,  # conservative GIC threshold
  IC                         = "GIC",
  plot                       = FALSE,
  store_model_fit_history    = FALSE,
  verbose                    = FALSE
)

res$shift_nodes_no_uncertainty
res$optimal_ic - res$baseline_ic
str(res$VCVs)

## Not run:
# Intercept-only empirical-style search:
# high-dimensional response matrix with H&L + measurement error
res_hl <- searchOptimalConfiguration(
  baseline_tree              = as.phylo(simmap),
  trait_data                 = X,
  formula                    = "trait_data ~ 1",
  min_descendant_tips        = 10,
  num_cores                  = 2,
  shift_acceptance_threshold = 20,
  uncertaintyweights_par     = TRUE,
  IC                         = "GIC",
  plot                       = FALSE,
  method                     = "H&L",
  error                      = TRUE,
  store_model_fit_history    = TRUE,
```

```
  verbose                    = TRUE
)

# Formula-based search with a predictor:
# use LL when the model includes predictors
dat <- data.frame(
  trait1    = X[, 1],
  trait2    = X[, 2],
  predictor = rnorm(nrow(X))
)
rownames(dat) <- simmap$tip.label

res_ll <- searchOptimalConfiguration(
  baseline_tree              = as.phylo(simmap),
  trait_data                 = dat,
  formula                    = "trait_data[, 1:2] ~ trait_data[, 3]",
  min_descendant_tips        = 10,
  num_cores                  = 2,
  shift_acceptance_threshold = 20,
  IC                         = "GIC",
  plot                       = FALSE,
  method                     = "LL",
  error                      = TRUE,
  store_model_fit_history    = TRUE,
  verbose                    = TRUE
)

## End(Not run)
```

Index