Package 'bifrost'

April 17, 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

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, phytools, grDevices, stats, mvMORPH, viridis, txtplot

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

VignetteBuilder knitr

Config/testthat/edition 3

Config/testthat/parallel false

Language en-US

NeedsCompilation no

Author Jacob S. Berv [aut, cre, cph, fnd] (ORCID:
<https://orcid.org/0000-0002-5962-0621>),
Nathan Fox [aut] (ORCID: <https://orcid.org/0000-0002-2816-9751>),
Matt J. Thorstensen [aut] (ORCID:
<https://orcid.org/0000-0002-7870-3369>),
Henry Lloyd-Laney [aut] (ORCID:
<https://orcid.org/0000-0003-4650-8937>),
Emily M. Troyer [aut] (ORCID: <https://orcid.org/0000-0001-7478-2306>),
Rafael A. Rivero-Vega [aut] (ORCID:
<https://orcid.org/0000-0001-5937-6377>),
Stephen A. Smith [aut, fnd] (ORCID:
<https://orcid.org/0000-0003-2035-9531>),
Matt Friedman [aut, fnd] (ORCID:
<https://orcid.org/0000-0002-0114-7384>),
David F. Fouhey [aut, fnd] (ORCID:
<https://orcid.org/0000-0001-5028-5161>),
Brian C. Weeks [aut, fnd] (ORCID:
<https://orcid.org/0000-0003-2967-2970>)

Maintainer Jacob S. Berv <jacob.berv@gmail.com>

Repository CRAN

Date/Publication 2026-04-17 04:00:03 UTC

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_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.

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 so that the multivariate response is interpreted correctly by `mvgls()`. When using more general formulas (e.g., pGLS-style models), a `data.frame` with named columns can be used instead.

`formula` Character formula 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, or `"trait_data[, 1:5] ~ trait_data[, 6]"` to fit a multivariate pGLS with column 6 as a predictor.

`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 parallel candidate scoring. 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 (Berv et al., in preparation) 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$tip.label` in both names and order; any tips without data should be pruned beforehand.
- *Data type:* `trait_data` is typically a numeric matrix of continuous traits; high-dimensional settings ($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 ΔIC $\geq$ `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. Candidate sub-model fits are distributed with **future** + **future.apply**. On Unix, `multicore` is used; on Windows, `multisession`. 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