Package 'stCEG'

July 7, 2025

Title Fully Customizable Chain Event Graphs over Spatial Areas

Version 0.1.0

Description Enables the creation of Chain Event Graphs over spatial areas, with an optional 'Shiny' user interface. Allows users to fully customise both the structure and underlying model of the Chain Event Graph, offering a high degree of flexibility for tailored analyses. For more details on Chain Event Graphs, see Freeman, G., & Smith, J. Q. (2011) <doi:10.1016/j.jmva.2011.03.008>, Collazo R. A., Görgen C. and Smith J. Q. (2018, ISBN:9781498729604) and Barclay, L. M., Hutton, J. L., & Smith, J. Q. (2014) <doi:10.1214/13-BA843>.

URL https://github.com/holliecalley/stCEG

BugReports https://github.com/holliecalley/stCEG/issues

License GPL (>= 3)

Encoding UTF-8

RoxygenNote 7.3.2

Imports DT, RColorBrewer, colorspace, crayon, dplyr, gtools, htmltools, htmlwidgets, hwep, igraph, leaflet, purrr, scales, sf, shiny, shinyWidgets, shinycssloaders, shinyjqui, shinyjs, sortable, spData, stringr, tidyr, tidyverse, viridis, visNetwork, zoo

Suggests randomcoloR, colourpicker

Depends R (>= 3.5)

LazyData true

NeedsCompilation no

Author Hollie Calley [aut, cre] (ORCID: <https://orcid.org/0009-0002-1814-8428>), Daniel Williamson [ctb] (ORCID: <https://orcid.org/0000-0001-8917-3300>)

Maintainer Hollie Calley <hc629@exeter.ac.uk>

Repository CRAN

Date/Publication 2025-07-07 13:00:02 UTC

Contents

`ahc_colouring` *Agglomerative Hierarchical Clustering (AHC) Colouring for Event Trees*

Description

This function applies Agglomerative Hierarchical Clustering (AHC) to colour the nodes of an event tree, taking into account their outgoing edges and stage information. It returns a coloured event tree based on the computed priors.

Usage

```
ahc_colouring(event_tree_obj, level_separation = 1000, node_distance = 300)
```

Arguments

`event_tree_obj`	A list containing an event tree or staged tree, and other relevant data for the event tree processing.
`level_separation`	Numeric value defining the level separation between nodes in the event tree (default is 1000).
`node_distance`	Numeric value defining the distance between nodes (default is 300).

Details

This function processes an event tree or partial staged tree, calculates priors based on the outgoing edges from each node, and performs Agglomerative Hierarchical Clustering (AHC) to colour the nodes of the event tree. It returns a `visNetwork` object that can be visualized as a coloured event tree. The function also computes the likelihood and scores based on merging stages in the event tree.

Value

A `visNetwork` object representing the staged tree.

Examples

```
data <- homicides
event_tree <- create_event_tree(data, columns = c(1,2,4,5), "both")
event_tree
coloured_tree <- ahc_colouring(event_tree)
coloured_tree
```

---

bcu_shapefile *London Basic Command Units (BCUs)*

Description

A simple features (`sf`) object containing the spatial boundaries of London's Basic Command Units (BCUs), used for policing. The dataset includes BCU names and their corresponding MULTIPOLYGON geometries. The data is projected in the British National Grid (EPSG:27700).

Usage

```
bcu_shapefile
```

Format

A `sf` object with 13 features and 1 field:

BCU Name of the Basic Command Unit (character)

geometry MULTIPOLYGON geometry column in BNG projection

Source

Merged from Borough shapefile from London Datastore.

Examples

```
library(sf)
plot(st_geometry(bcu_shapefile))
```

borough_shapefile *London Borough Boundaries*

Description

A sf object containing the spatial boundaries of London boroughs, projected in the British National Grid (EPSG:27700). This dataset includes the borough names and associated MULTIPOLYGON geometries.

Usage

```
borough_shapefile
```

Format

A sf object with 33 features and 1 field:

Borough Name of the London Borough (character)

geometry MULTIPOLYGON geometry column in BNG projection

Source

London Datastore

Examples

```
library(sf)
plot(st_geometry(borough_shapefile))
```

calculate_area_probabilities *Calculate Area Probabilities in a Chain Event Graph (CEG)*

Description

This function calculates the conditional probability of a given last group for each area, based on the path products and area-specific paths. It leverages the calculate_conditional_prob function for each area.

Usage

```
calculate_area_probabilities(
  path_df,
  unique_values,
  selected_indices,
  last_group,
  shapefile_vals
)
```

Arguments

path_df	A data frame containing paths and their corresponding products, as returned by the calculate_path_products function.
unique_values	A character vector containing the unique values (e.g., labels or categories) to condition on.
selected_indices	A numeric vector of indices corresponding to the groupings of the unique values.
last_group	A string representing the last group for which the conditional probability is to be calculated.
shapefile_vals	A vector of area names (or other geographical identifiers) from the shapefile data.

Value

A list with area names as keys and their respective conditional probabilities as values. If no paths are found for a specific area, the probability is returned as NA.

---

calculate_conditional_prob

Calculate Conditional Probability in a Chain Event Graph (CEG)

Description

This function calculates the conditional probability of a specified group of conditions occurring, given a set of unique values, based on the calculated path products.

Usage

```
calculate_conditional_prob(
  path_df,
  unique_values,
  selected_indices,
  last_group
)
```

Arguments

path_df	A data frame containing paths and their corresponding products, as returned by the calculate_path_products function.
unique_values	A character vector containing the unique values (e.g., labels or categories) to condition on.
selected_indices	A numeric vector of indices corresponding to the groupings of the unique values.
last_group	A string representing the last group for which the conditional probability is to be calculated.

Value

A numeric value representing the conditional probability P(last_group | unique_values).

---

calculate_path_products

Calculate Path Products in a Chain Event Graph (CEG)

Description

This function calculates the products of probabilities along all paths from a specified root node in a Chain Event Graph (CEG). It traverses the graph, multiplying the posterior means of the edges for each path, and returns the resulting paths and their products.

Usage

```
calculate_path_products(nodes_df, edges_df, root_node = "w0")
```

Arguments

nodes_df	A data frame representing the nodes in the CEG.
edges_df	A data frame representing the edges in the CEG, containing columns for from, to, label1, and posterior_mean.
root_node	A string specifying the label of the root node from which to start the traversal. Default is "w0".

Value

A data frame with two columns:

path	The sequence of nodes in the path, as a string of node labels separated by " -> "
product	The product of the posterior means along the path

`compare_ceg_models` *Compare Two Chain Event Graph Models Using Bayes Factors*

Description

This function compares two fitted Chain Event Graph (CEG) models by evaluating their total log marginal likelihoods and computing the Bayes factor. The comparison identifies the preferred model and reports the strength of evidence using Jeffreys' scale.

Usage

```
compare_ceg_models(summary1, summary2)
```

Arguments

`summary1`	An object of class `summary.chain_event_graph`, typically the output of `summary()` applied to a fitted CEG model.
`summary2`	Another object of class `summary.chain_event_graph` to compare with `summary1`.

Details

The Bayes factor is calculated as the ratio of marginal likelihoods of the two models: $BF = \exp(\log BF)$. Interpretation of the Bayes factor is guided by Jeffreys' scale:

- **< 1:** Evidence against the alternative model
- **1–3:** Weak evidence
- **3–10:** Moderate evidence
- **10–30:** Strong evidence
- **30–100:** Very strong evidence
- **> 100:** Decisive evidence

The function prints the log marginal likelihoods, log Bayes factor, Bayes factor, and preferred model.

Value

An invisible list of class `ceg_model_comparison` containing:

log_marginal_1 Log marginal likelihood of model 1.

log_marginal_2 Log marginal likelihood of model 2.

log_Bayes_factor The log Bayes factor comparing model 1 to model 2.

Bayes_factor The Bayes factor (on the original scale).

preferred_model The model preferred based on the Bayes factor.

Examples

```
data <- homicides
event_tree <- create_event_tree(data, columns = c(1,2,4,5), "both")
coloured_tree <- ahc_colouring(event_tree)

tree_priors <- specify_priors(coloured_tree, prior_type = "Uniform", ask_edit = FALSE)
staged_tree <- staged_tree_prior(coloured_tree, tree_priors)
ceg <- create_ceg(staged_tree, view_table = TRUE)


# Define node groups and colours
node_groups <- list(c("s1", "s2"), c("s3", "s4"))
colours <- c("#BBA0CA", "#8AC6D0")

# Apply colours to the event tree
custom_tree <- update_node_colours(event_tree, node_groups, colours)
custom_coloured_tree <- ahc_colouring(custom_tree)

# Cannot run this whole chunk at once as specify_priors needs user input
custom_tree_priors <- specify_priors(custom_coloured_tree, prior_type = "Uniform", ask_edit = FALSE)
custom_staged_tree <- staged_tree_prior(custom_coloured_tree, custom_tree_priors)
ceg2 <- create_ceg(custom_staged_tree, view_table = TRUE)
model1_summary <- summary(ceg)
model2_summary <- summary(ceg2)
compare_ceg_models(model1_summary, model2_summary)
```

---

create_ceg *Create a Chain Event Graph (CEG)*

Description

This function generates a Chain Event Graph (CEG) from a staged tree object and a prior table. It allows for node contraction and mapping, adjusts edge and node attributes, and visualizes the graph with specific customisations for labels, colour, and node arrangement.

Usage

```
create_ceg(
  staged_tree_obj,
  level_separation = 1200,
  node_distance = 400,
  label = "posterior",
  view_table = FALSE
)
```

Arguments

`staged_tree_obj`
: A staged tree object containing nodes and edges. It should have the following structure:
  - `staged_tree_obj$x$nodes` : A data frame of node attributes with columns such as `id`, `level`, `colour`, `label`, etc.
  - `staged_tree_obj$x$edges` : A data frame of edge attributes with columns such as `from`, `to`, `label1`, `label2`, `label3`, etc.

`level_separation`
: Numeric. The level separation value for hierarchical layout in the visualised graph. Default is 1200.

`node_distance`
: Numeric. The node distance value for hierarchical layout in the visualised graph. Default is 400.

`label`
: A character string specifying the type of label to display on edges. Options include:
  - `"prior"`: Uses posterior label information.
  - `"prior_mean"`: Uses posterior label information.
  - `"posterior"`: Uses posterior label information.
  - `"posterior_mean"`: Uses posterior mean label information.
  - `"none"`: No labels on edges. Default is `"posterior_mean"`.

`view_table`
: Logical. Whether to display the summary table of the aggregated CEG data in the console. Default is `FALSE`.

Details

This function processes the staged tree and prior table, contracts nodes based on connected nodes, creates aggregated edge summaries, computes posterior and prior mean values, and visualizes the CEG with hierarchical layout, customizable labels, and node distance adjustments.

The function also provides detailed printing of updated edges and contracted nodes for debugging purposes.

Value

A `visNetwork` object representing the Chain Event Graph, including contracted nodes and updated edges.If view_table = TRUE, a coloured table is returned that can be displayed in the viewer, instead of just an update_table which can be viewed in the console.

Examples

```
data <- homicides
event_tree <- create_event_tree(data, columns = c(1,2,4,5), "both")
coloured_tree <- ahc_colouring(event_tree)

# Cannot run this whole chunk at once as specify_priors needs user input
tree_priors <- specify_priors(coloured_tree, prior_type = "Uniform", ask_edit = FALSE)
staged_tree <- staged_tree_prior(coloured_tree, tree_priors)
ceg <- create_ceg(staged_tree, view_table = TRUE)
```

`create_event_tree` *Create an Event Tree from a Dataset*

Description

This function constructs an event tree from a given dataset based on user-specified categorical variables. The resulting tree is represented as a graph, where each unique combination of variable values corresponds to a distinct path in the tree.

Usage

```
create_event_tree(
  dataset,
  columns = seq_along(dataset),
  label_type = "both",
  level_separation = 1000,
  node_distance = 300
)
```

Arguments

`dataset`	A data frame containing categorical variables.
`columns`	A vector of column indices or names specifying which variables to use for constructing the event tree.
`label_type`	A character string specifying how edge labels should be displayed. Options are `"names"` (variable names), `"both"` (variable names and counts). Default is `"both"`.
`level_separation`	A numeric value indicating the spacing between levels in the visualization. Default is `1000`.
`node_distance`	A numeric value specifying the horizontal distance between nodes in the visualization. Default is `300`.

Details

The function follows these steps:

1. Extracts the specified columns from the dataset.
2. Determines unique values for each variable and generates state names dynamically.
3. Constructs a directed acyclic graph (DAG) using `igraph`, where each state represents a unique combination of variable values.
4. Computes transition counts between states.
5. Uses `visNetwork` to generate an interactive visualization of the event tree.

Value

A visNetwork object displaying the event tree.

Examples

```
data <- homicides
event_tree <- create_event_tree(data, columns = c(1,2,4,5), "both")
event_tree
```

---

create_reduced_CEG *Create a Reduced Chain Event Graph (CEG)*

Description

This function generates a reduced version of a Chain Event Graph (CEG) based on a set of starting labels. It extracts connected nodes and edges from the input CEG object, and visualizes the result using the visNetwork package.

Usage

```
create_reduced_CEG(
  ceg_object,
  start_labels,
  level_separation = 1200,
  node_distance = 300
)
```

Arguments

ceg_object	A list containing the Chain Event Graph (CEG) data, including nodes and edges. The ceg_object must have the structure ceg_objectcegx$nodes and ceg_object$ceg$x$edges.
start_labels	A character vector containing the labels of the nodes from which the floret extraction should start.
level_separation	A numeric value determining the separation between levels in the hierarchical layout. Default is 1200.
node_distance	A numeric value controlling the distance between nodes in the hierarchical layout. Default is 300.

Value

A visNetwork object representing the reduced CEG, with interactive visualization features, such as node selection, highlighting of edges, and layout manipulation.

Examples

```
data <- homicides
event_tree <- create_event_tree(data, columns = c(1,2,4,5), "both")
coloured_tree <- ahc_colouring(event_tree)

tree_priors <- specify_priors(coloured_tree, prior_type = "Uniform", ask_edit = FALSE)
staged_tree <- staged_tree_prior(coloured_tree, tree_priors)
ceg <- create_ceg(staged_tree, view_table = TRUE)
create_reduced_CEG(ceg, "Adult")
```

---

delete_nodes *Delete Nodes from an Event Tree*

Description

This function removes specified nodes from an event tree, updating edges to maintain the tree structure while ensuring that node IDs remain sequential.

Usage

```
delete_nodes(
  event_tree_obj,
  nodes_to_delete,
  level_separation = 1000,
  node_distance = 300
)
```

Arguments

`event_tree_obj`	A list containing the event tree object, which includes: • `$eventtree$x`: A list with `nodes` and `edges` data frames. • `$filtereddf`: The data frame used to create the event tree.
`nodes_to_delete`	A character vector of node IDs to delete from the event tree.
`level_separation`	Numeric value specifying the spacing between levels in the hierarchical layout. Default is `1000`.
`node_distance`	Numeric value specifying the distance between nodes in the layout. Default is `300`.

Details

The function performs the following steps:

- Identifies outgoing and incoming edges for each node marked for deletion.
- Redirects outgoing edges to the sources of the incoming edges.
- Updates the edges and nodes data frames to reflect the new connections.
- Ensures node IDs are sequentially re-assigned.
- Adjusts outgoing edge counts for affected nodes.
- Removes orphaned nodes (nodes with no connections).
- Returns an updated `visNetwork` visualization of the event tree.

Value

A list containing:

- `$eventtree`: A `visNetwork` object representing the updated event tree.
- `$filtereddf`: The filtered data frame, returned invisibly.

Examples

```
data <- homicides
event_tree <- create_event_tree(data, columns = c(1,2,4,5), "both")
event_tree

updated_tree <- delete_nodes(event_tree, nodes_to_delete = c("s14", "s18"))
updated_tree
```

---

generate_CEG_map *Generate a Leaflet Map for a Chain Event Graph (CEG)*

Description

This function generates an interactive map using the Leaflet package to visualize the probability of each area in the Chain Event Graph (CEG). The map is color-coded based on area-specific probabilities, which are calculated from the path products and conditional probabilities.

Usage

```
generate_CEG_map(
  shapefile,
  ceg_object,
  conditionals = unique(ceg_object$x$edges$label1),
  colour_by = NULL,
  color_palette = "viridis"
)
```

Arguments

shapefile	A Simple Features (sf) object representing the shapefile data for the geographical areas.
ceg_object	A list containing the Chain Event Graph (CEG) data, including nodes and edges.
conditionals	A character vector containing the conditions (labels) to condition on when calculating area probabilities. Default is the unique edge labels from the ceg_object.
colour_by	A string specifying the label by which to color the map. Default is NULL, which colors by the label with the maximum level.
color_palette	A string specifying the color palette to use for the map. Default is "viridis".

Value

A Leaflet map object with color-coded polygons representing the areas, with a legend indicating the probability values for each area.

Examples

```
data <- homicides
event_tree <- create_event_tree(data, columns = c(9,2,4,5), "both")
coloured_tree <- ahc_colouring(event_tree)

tree_priors <- specify_priors(coloured_tree, prior_type = "Uniform", ask_edit = FALSE)
staged_tree <- staged_tree_prior(coloured_tree, tree_priors)
ceg <- create_ceg(staged_tree, view_table = TRUE)
generate_CEG_map(bcu_shapefile, ceg)
```

---

homicides *Homicides Dataset*

Description

A dataset containing homicides recorded by the Metropolitan Police from 2003-2023

Usage

```
homicides
```

Format

A data frame with 2670 rows and 9 variables:

Age_Group Age group of the victim. One of: "Adult", "Adolescent/Young Adult", "Child", "Elderly".

Sex Sex of the victim. "Male" or "Female"

Method_of_Killing Recorded method of killing, e.g., "Knife or Sharp Implement", "Blunt Implement", etc.

Domestic_Abuse Indicates whether the incident was flagged as domestic abuse.

Solved_Status Whether the case has been solved. One of: `"Solved"`, `"Unsolved"`

Borough London Borough where the homicide occurred

Ethnicity Recorded ethnicity of the victim. One of `"Asian"`, `"Black"`, `"Not Reported/Not Known"`, `"Other"`, `"White"`.

Year Year in which the incident was recorded.

BCU Basic Command Unit where the homicide occurred (Homicide units changed from Boroughs to BCUs in 2017).

Source

Data taken and filtered from https://www.met.police.uk/police-forces/metropolitan-police/areas/stats-and-data/stats-and-data/met/homicide-dashboard/

---

run_stceg *Run the stCEG Shiny App*

Description

The app facilitates the upload and processing of spatial data, including the dynamic selection of area divisions, time divisions, and prediction variables. It integrates the leaflet package for interactive maps and visNetwork for visualizing event trees. Additionally, the app supports the use of agglomerative hierarchical clustering (AHC) Run the stCEG Shiny App

This function launches a Shiny app that allows users to upload spatial data (in the form of a shapefile) and visualize it using Chain Event Graphs (CEGs). The app enables the user to apply various spatial and temporal analyses, including map visualizations and event tree colourings.

The app facilitates the upload and processing of spatial data, including the dynamic selection of area divisions, time divisions, and prediction variables. It integrates the leaflet package for interactive maps and visNetwork for visualizing event trees. Additionally, the app supports the use of agglomerative hierarchical clustering (AHC) for node colouring in event trees and allows for specification of prior distributions.

Usage

```
run_stceg()
```

Details

This function launches a Shiny app that allows users to upload spatial data (in the form of a shapefile) and visualize it using Chain Event Graphs (CEGs). The app enables the user to apply various spatial and temporal analyses, including map visualizations and event tree colourings.

Value

The function launches a Shiny app for spatial data analysis with interactive visualizations and data manipulation features.

Examples

```
if(interactive()){
# Run the app
run_stceg()}
```

`specify_priors` *Specify Priors for a Staged Tree Object*

Description

This function assigns priors to the nodes of a staged tree object. The user can choose from different types of priors, including "Uniform", "Phantom", and "Custom".

Usage

```
specify_priors(
  staged_tree_obj,
  prior_type = "Uniform",
  ask_edit = TRUE,
  print_colours = TRUE
)
```

Arguments

`staged_tree_obj`
: A staged tree object containing the nodes and edges data for the tree.

`prior_type`
: A character string indicating the prior type. Options are:
  - "Uniform": Assigns a Uniform (1,1) prior based on the outgoing edges.
  - "Phantom": Calculates a Phantom Individuals prior by initialising an alpha based on the maximum number of outgoing edges in the tree and dividing that evenly throughout the tree.
  - "Custom": Allows the user to manually specify the prior values for each node.

`ask_edit`
: If TRUE, this allows you to edit priors after choosing an uninformative prior type.

`print_colours`
: If TRUE, this prints a colour key for stages. Default is TRUE.

Details

The function checks if the necessary columns and structure are present in the input staged tree object. It performs validation on the nodes' levels and ensures that no nodes with non-minimum or non-maximum levels have the colour `#FFFFFF`. For the "Uniform" and "Phantom" priors, the function calculates the priors for each node based on their outgoing edges and propagates them through connected nodes. The user can edit these priors if desired. The "Custom" option allows the user to manually input prior values for each row. If incorrect values are provided, an error will be raised.

Value

A data frame containing the updated nodes data with the specified priors and their means.

Examples

```
data <- homicides
event_tree <- create_event_tree(data, columns = c(1,2,4,5), "both")
coloured_tree <- ahc_colouring(event_tree)
tree_priors <- specify_priors(coloured_tree, prior_type = "Uniform", ask_edit = FALSE)
```

---

staged_tree_prior *Apply Priors to a Staged Tree for Visualization*

Description

This function applies prior distributions to a staged tree, adjusting priors and calculating the mean and variance of the Dirichlet distributions for each node. The function also generates a visualization using the `visNetwork` package, showing the tree with additional information like prior distributions, means, and variances.

Usage

```
staged_tree_prior(
  staged_tree_obj,
  prior_table,
  level_separation = 1500,
  node_distance = 250,
  label_type = "priors"
)
```

Arguments

`staged_tree_obj`	A list containing a staged tree object with the components "nodes" and "edges".
`prior_table`	A data frame with columns "Colour", "Level", and "Prior", containing prior distributions for each stage.
`level_separation`	A numeric value determining the separation between levels in the hierarchical layout of the tree. Default is 1000.
`node_distance`	A numeric value for the distance between nodes. Default is 300.
`label_type`	A string indicating the label type for edges. It can be one of "names", "priors", or "priormeans". Default is "names".

Details

- The function assumes that the staged tree object is structured in a specific format, where the nodes and edges are contained in the "stagedtree" component.
- Priors are adjusted based on the number of nodes in each group, and Dirichlet priors are split and normalized.
- Tooltips for nodes show the prior distribution, mean, and variance.
- Edge labels can show the names, priors, or prior means based on the `label_type` parameter.

Value

A `visNetwork` object that displays the staged tree with adjusted prior distributions and additional information in the tooltips.

Examples

```
data <- homicides
event_tree <- create_event_tree(data, columns = c(1,2,4,5), "both")
coloured_tree <- ahc_colouring(event_tree)
# cannot run this whole chunk as one, as specify_priors needs user input
tree_priors <- specify_priors(coloured_tree, prior_type = "Uniform", ask_edit = FALSE)
staged_tree <- staged_tree_prior(coloured_tree, tree_priors)
staged_tree
```

`summary.chain_event_graph`

Summarise a Chain Event Graph Model

Description

Computes the total log marginal likelihood, effective sample size (ESS), and per-stage log scores for a fitted Chain Event Graph (CEG) model using conjugate prior/posterior updates.

Usage

```
## S3 method for class 'chain_event_graph'
summary(object, ...)
```

Arguments

`object`	An object of class `chain_event_graph`, which must contain an `update_table` with prior and data columns for each stage.
`...`	Additional arguments (currently unused).

Details

The log marginal likelihood is computed using the Dirichlet-multinomial formula.

The effective sample size for a stage is defined as $ESS = \sum_j (\alpha_{ij} + y_{ij})$. It represents the amount of information (prior + observed) available for that stage.

Value

An invisible object of class `summary.chain_event_graph`, which is a list containing:

total_log_marginal_likelihood Total log marginal likelihood across all stages.

per_stage_log_scores A data frame with log scores and effective sample sizes (ESS) for each stage.

Examples

```
data <- homicides
event_tree <- create_event_tree(data, columns = c(1,2,4,5), "both")
coloured_tree <- ahc_colouring(event_tree)

# Cannot run this whole chunk at once as specify_priors needs user input
tree_priors <- specify_priors(coloured_tree, prior_type = "Uniform", ask_edit = FALSE)
staged_tree <- staged_tree_prior(coloured_tree, tree_priors)
ceg <- create_ceg(staged_tree, view_table = TRUE)
homicides_CEG_summary <- summary(ceg)
```

`summary.event_tree` *Summary of an Event Tree*

Description

This function provides a summary of an object of class `event_tree`. It outputs key information about the nodes and edges of the event tree, including the number of nodes and edges, unique levels and labels, and the labels of the edges (with newlines replaced by spaces).

Usage

```
## S3 method for class 'event_tree'
summary(object, ...)
```

Arguments

`object`	An object of class `event_tree`. This object should have a list structure with `eventtree` containing a nested list where `x` is a dataframe with `nodes` and `edges` attributes.
`...`	Additional arguments passed to or from other methods (ignored).

Value

Prints a summary of the event tree to the console, including:

- The number of nodes and the unique node levels
- The number of edges and the unique labels in the edges
- The labels of the edges, with newlines replaced by spaces

Examples

```
data <- homicides
event_tree <- create_event_tree(data, columns = c(1,2,4,5), "both")
homicides_ET_summary <- summary(event_tree)
```

---

summary.staged_tree *Summarise a Staged Tree Object*

Description

Provides a visual and textual summary of a staged tree object, including counts of nodes and edges, node colour distributions, and uncoloured nodes in intermediate levels.

Usage

```
## S3 method for class 'staged_tree'
summary(object, ...)
```

Arguments

object	An object of class staged_tree, containing components nodes and edges in object$stagedtree$x, and optionally a priortable.
...	Additional arguments (currently unused).

Details

This function #prints a summary that includes:

- Total number of nodes and edges.
- Number of nodes still uncoloured (white) that are not in the first or last level.
- A count of nodes by hex colour, optionally with coloured terminal output using the crayon package.
- The prior table if it exists.

If the crayon package is available, the function displays background colour blocks in the console to represent node colours.

Value

Invisibly returns a list containing summary components:

- num_nodes: Total number of nodes in the staged tree.
- num_edges: Total number of edges in the staged tree.
- num_uncoloured_middle: Number of white nodes not in the first or last level.
- color_counts: A named table of node colour frequencies.
- priortable: The prior table, if available; otherwise NULL.

Also prints a human-readable summary to the console.

Note

White-coloured nodes (#FFFFFF) that are not in the minimum or maximum level are flagged as "Nodes left to be coloured."

Examples

```
data <- homicides
event_tree <- create_event_tree(data, columns = c(1,2,4,5), "both")
coloured_tree <- ahc_colouring(event_tree)
tree_priors <- specify_priors(coloured_tree, prior_type = "Uniform", ask_edit = FALSE)
staged_tree <- staged_tree_prior(coloured_tree, tree_priors)
homicides_ST_summary <- summary(staged_tree)
```

---

update_node_colours *Update Node Colours in an Event Tree*

Description

This function updates the colours of nodes in an event tree based on user-defined groups while ensuring that nodes with the same colour have consistent outgoing edge labels.

Usage

```
update_node_colours(
  event_tree_obj,
  node_groups,
  colours,
  level_separation = 1000,
  node_distance = 300
)
```

Arguments

event_tree_obj	A list containing the event tree object (possibly with node deletion). It should include: • $eventtree$x$nodes: A data frame of nodes. • $eventtree$x$edges: A data frame of edges. • $filtereddf: A filtered data frame associated with the event tree.
node_groups	A list of character vectors, where each vector contains node IDs belonging to a specific group.
colours	A character vector of colour codes corresponding to each node group. The length of colours must match node_groups.
level_separation	Numeric value specifying the spacing between levels in the hierarchical layout. Default is 1000.
node_distance	Numeric value specifying the distance between nodes in the layout. Default is 300.

Details

The function follows these steps:

- Assigns colours to nodes based on the specified node_groups.
- Ensures that no node appears in multiple groups (raises an error if duplicates exist).
- Checks that all nodes with the same colour have identical outgoing edge labels.
- Updates the event tree visualization using visNetwork.

Value

A list containing:

- $stagedtree: A visNetwork object representing the updated event tree with coloured nodes.
- $filtereddf: The filtered data frame, returned invisibly.

Examples

```
data <- homicides
event_tree <- create_event_tree(data, columns = c(1,2,4,5), "both")
event_tree

# Define node groups and colours
node_groups <- list(c("s1", "s2"), c("s3", "s4"))
colours <- c("#BBA0CA", "#8AC6D0")

# Apply colours to the event tree
coloured_tree <- update_node_colours(event_tree, node_groups, colours)
coloured_tree
```

Index