

Package ‘Nestimate’

July 11, 2026

Title Network Estimation, Bootstrap, and Higher-Order Analysis

Version 0.8.0

Description Estimate, compare, and analyze dynamic and psychological networks using a unified interface. Provides transition network analysis estimation (transition, frequency, co-occurrence, attention-weighted) Saqr et al. (2025) <[doi:10.1145/3706468.3706513](https://doi.org/10.1145/3706468.3706513)>, psychological network methods (correlation, partial correlation, 'graphical lasso', 'Ising') Saqr, Beck, and Lopez-Pernas (2024) <[doi:10.1007/978-3-031-54464-4_19](https://doi.org/10.1007/978-3-031-54464-4_19)>, and higher-order network methods including higher-order networks, higher-order network embedding, hyper-path anomaly, and multi-order generative model. Supports bootstrap inference, permutation testing, split-half reliability, centrality stability analysis, mixed Markov models, multi-cluster multi-layer networks and clustering.

License MIT + file LICENSE

URL <https://github.com/mohsaqr/Nestimate>, <https://saqr.me/Nestimate/>

BugReports <https://github.com/mohsaqr/Nestimate/issues>

Language en-US

Encoding UTF-8

RoxygenNote 7.3.3

Imports ggplot2, data.table, cluster, scales, brglm2, nnet

Suggests testthat (>= 3.0.0), igraph, glmnet, lavaan, stringdist, gridExtra, lme4, corpcor, Matrix, cograph (>= 2.4.4), ggfittext, knitr, rmarkdown, tna

Config/testthat/edition 3

Depends R (>= 4.1.0)

LazyData true

VignetteBuilder knitr

NeedsCompilation no

Author Mohammed Saqr [aut, cre, cph],
 Sonsoles López-Pernas [aut],
 Kamila Misiejuk [aut]
Maintainer Mohammed Saqr <saqr@saqr.me>
Repository CRAN
Date/Publication 2026-07-10 23:10:22 UTC

Contents

action_to_onehot	7
actor_endpoints	8
association_rules	9
as_htna	11
as_netdifference	13
as_netobject	14
as_networks	15
as_tna	16
bayes_compare	18
betti_numbers	20
bipartite_groups	21
bootstrap_network	22
boot_glasso	24
bottleneck_distance	27
build_atna	28
build_clusters	29
build_cna	32
build_cor	33
build_ftna	33
build_gimme	34
build_glasso	37
build_hon	37
build_honem	40
build_hypa	41
build_hypergraph	43
build_ising	45
build_mcml	46
build_mcml_pc	49
build_mlvar	54
build_mmm	56
build_mogen	58
build_network	60
build_pcor	64
build_simplicial	65
build_tna	66
casedrop_reliability	67
centrality_stability	70
certainty	72

chain_structure	74
chatgpt_srl	76
clique_expansion	77
cluster_choice	78
cluster_diagnostics	79
cluster_mmm	81
cluster_network	83
cluster_summary	84
coefs	89
compare_mmm	90
compare_model	91
compare_model.netobject_group	93
convert_sequence_format	94
cooccurrence	95
distribution_plot	98
estimate_network	100
euler_characteristic	101
evaluate_links	102
extract_edges	103
extract_initial_probs	104
extract_transition_matrix	105
frequencies	106
get_estimator	107
group_regulation_long	108
hypergraph_centrality	109
hypergraph_measures	110
learning_activities	112
list_estimators	113
loading_stability	114
long-data	115
long_to_wide	116
magnitude_difference	117
markov_order_test	119
markov_stability	121
mark_first_state	122
mark_terminal_state	123
mogen_transitions	124
mosaic_analysis	125
mosaic_plot	127
nct	130
network_reliability	132
net_aggregate_weights	133
net_centrality	134
net_deprune	135
net_edge_betweenness	136
net_prune	138
net_pruning_details	140
net_reprune	141

passage_time	142
pathways	143
path_counts	146
path_dependence	147
permutation	148
persistence_landscape	151
persistent_homology	152
plot.boot_glasso	153
plot.chain_structure	154
plot.cluster_choice	155
plot.mcml_pc	156
plot.mmm_compare	156
plot.mosaic_analysis	157
plot.net_association_rules	158
plot.net_bayes	159
plot.net_centrality	159
plot.net_centrality_group	160
plot.net_clustering	162
plot.net_cluster_diagnostics	163
plot.net_comparison	163
plot.net_edge_betweenness	164
plot.net_gimme	165
plot.net_honem	166
plot.net_markov_order	167
plot.net_mmm	168
plot.net_mmm_clustering	169
plot.net_mogen	170
plot.net_path_dependence	171
plot.net_reliability	171
plot.net_sequence_comparison	172
plot.net_stability	173
plot.net_transition_entropy	174
plot.net_vertex_bootstrap	175
plot.net_vertex_comparison	175
plot.pc_loading_stability	176
plot.persistence_landscape	176
plot.persistent_homology	177
plot.q_analysis	178
plot.simplicial_complex	179
plot_mosaic	179
plot_state_frequencies	181
predictability	185
predict_links	186
prepare	188
prepare_for_tna	190
prepare_onehot	191
print.boot_glasso	193
print.chain_structure	193

print.chain_structure_group	194
print.cluster_choice	194
print.mcml	195
print.mcml_layer	196
print.mcml_pc	196
print.mmm_compare	197
print.mosaic_analysis	197
print.nestimate_data	198
print.netobject	199
print.netobject_group	199
print.netobject_ml	200
print.net_association_rules	201
print.net_bayes	202
print.net_bayes_group	203
print.net_bootstrap	203
print.net_bootstrap_group	204
print.net_certainty	205
print.net_clustering	206
print.net_cluster_diagnostics	207
print.net_gimme	207
print.net_hon	208
print.net_honem	209
print.net_hypa	209
print.net_link_prediction	210
print.net_markov_order	211
print.net_markov_order_group	212
print.net_markov_stability_group	212
print.net_mlvar	213
print.net_mmm	214
print.net_mmm_clustering	215
print.net_mogen	215
print.net_mpt_group	216
print.net_nct	217
print.net_path_dependence	217
print.net_permutation	218
print.net_permutation_group	219
print.net_pruning_details	220
print.net_reliability	220
print.net_sequence_comparison	221
print.net_stability	222
print.net_stability_group	223
print.net_transition_entropy	223
print.net_transition_entropy_group	224
print.net_vertex_bootstrap	224
print.net_vertex_comparison	225
print.pc_loading_stability	225
print.persistence_landscape	226
print.persistent_homology	226

print.q_analysis	227
print.simplicial_complex	228
print.summary.net_path_dependence	228
print.summary.net_transition_entropy	229
print.summary_chain_structure	229
print.tidy_covariates	230
print.wtna_boot_mixed	230
print.wtna_mixed	231
print.wtna_perm_mixed	232
q_analysis	233
register_estimator	233
remove_estimator	234
rename_models	235
sequence_compare	236
sequence_plot	237
simplicial_degree	241
srl_strategies	242
state_distribution	242
state_freq	243
state_frequencies	244
subtract_networks	245
summary.boot_glasso	246
summary.chain_structure	247
summary.chain_structure_group	247
summary.cluster_choice	248
summary.mcml	248
summary.mcml_pc	249
summary.mmm_compare	250
summary.mosaic_analysis	250
summary.nest_initial_probs	251
summary.nest_transition_counts	251
summary.nest_transition_matrix	252
summary.netobject	252
summary.netobject_group	253
summary.net_association_rules	253
summary.net_bayes	254
summary.net_bayes_group	255
summary.net_bootstrap	255
summary.net_bootstrap_group	256
summary.net_clustering	257
summary.net_gimme	258
summary.net_hon	259
summary.net_honem	260
summary.net_hypa	260
summary.net_link_prediction	262
summary.net_markov_order	262
summary.net_mlvar	263
summary.net_mmm	264

summary.net_mogen	265
summary.net_nct	266
summary.net_path_dependence	267
summary.net_permutation	267
summary.net_permutation_group	268
summary.net_reliability	269
summary.net_sequence_comparison	270
summary.net_stability	271
summary.net_stability_group	272
summary.net_transition_entropy	272
summary.net_vertex_bootstrap	273
summary.net_vertex_comparison	273
summary.wtna_boot_mixed	274
summary.wtna_perm_mixed	275
trajectories	275
transition_entropy	276
validate_netobject	277
verify_simplicial	278
vertex_bootstrap	279
vertex_compare	281
wide_to_long	283
wtna	284
Index	287

action_to_onehot	<i>Convert Action Column to One-Hot Encoding</i>
------------------	--

Description

Convert a categorical Action column to one-hot (binary indicator) columns.

Usage

```
action_to_onehot(  
  data,  
  action_col = "Action",  
  states = NULL,  
  drop_action = TRUE,  
  sort_states = FALSE,  
  prefix = ""  
)
```

Arguments

<code>data</code>	Data frame containing an action column.
<code>action_col</code>	Character. Name of the action column. Default: "Action".
<code>states</code>	Character vector or NULL. States to include as columns. If NULL, uses all unique values. Default: NULL.
<code>drop_action</code>	Logical. Remove the original action column. Default: TRUE.
<code>sort_states</code>	Logical. Sort state columns alphabetically. Default: FALSE.
<code>prefix</code>	Character. Prefix for state column names. Default: "".

Value

Data frame with one-hot encoded columns (0/1 integers).

Examples

```
long_data <- data.frame(
  Actor = rep(1:3, each = 4),
  Time = rep(1:4, 3),
  Action = sample(c("A", "B", "C"), 12, replace = TRUE)
)
onehot_data <- action_to_onehot(long_data)
head(onehot_data)
```

<code>actor_endpoints</code>	<i>Tidy per-actor endpoint summary of a wide-format sequence dataset</i>
------------------------------	--

Description

For each actor (row), reports the first and last observed states, the time indices at which they appear, the number of observed steps, and a `dropped_out` flag that is TRUE when the actor has a *terminal-NA* pattern (after the final observed step, every remaining cell is NA).

Usage

```
actor_endpoints(data, cols = NULL)
```

Arguments

<code>data</code>	A wide-format matrix or <code>data.frame</code> where rows are actors and columns are time steps. Cells are state labels; NA represents missing observations. If <code>data</code> is a <code>data.frame</code> , non-character/-factor columns (e.g. an <code>id</code> column) are dropped via the <code>cols</code> argument.
<code>cols</code>	Optional character vector of state-column names. If NULL (default) every column is treated as a state column.

Value

A tidy data.frame with one row per actor and columns:

actor Row number (or row name if present).

first_state First non-NA state.

last_state Last non-NA state.

first_step Column index of the first observed state.

last_step Column index of the last observed state.

n_observed Number of non-NA cells.

dropped_out TRUE iff every cell after last_step is NA and last_step < ncol(data).

See Also

[mark_terminal_state\(\)](#), [chain_structure\(\)](#)

Examples

```
actor_endpoints(trajectories) |> head()
```

association_rules

Discover Association Rules from Sequential or Transaction Data

Description

Discovers association rules using the Apriori algorithm with proper candidate pruning. Accepts netobject (extracts sequences as transactions), data frames, lists, or binary matrices.

Support counting is vectorized via crossprod() for 2-itemsets and logical matrix indexing for k-itemsets.

Usage

```
association_rules(  
  x,  
  min_support = 0.1,  
  min_confidence = 0.5,  
  min_lift = 1,  
  max_length = 5L  
)
```

Arguments

x	Input data. Accepts: netobject Uses \$data sequences - each sequence becomes a transaction of its unique states. list Each element is a character vector of items (one transaction). data.frame Wide format: each row is a transaction, character columns are item occurrences. Or a binary matrix (0/1). matrix Binary transaction matrix (rows = transactions, columns = items).
min_support	Numeric. Minimum support threshold. Default: 0.1.
min_confidence	Numeric. Minimum confidence threshold. Default: 0.5.
min_lift	Numeric. Minimum lift threshold. Default: 1.0.
max_length	Integer. Maximum itemset size. Default: 5.

Details

Algorithm:

Uses level-wise Apriori (Agrawal & Srikant, 1994) with the full pruning step: after the join step generates k-candidates, all (k-1)-subsets are verified as frequent before support counting. This is critical for efficiency at $k \geq 4$.

Metrics:

support $P(A \text{ and } B)$. Fraction of transactions containing both antecedent and consequent.

confidence $P(B | A)$. Fraction of antecedent transactions that also contain the consequent.

lift $P(A \text{ and } B) / (P(A) * P(B))$. Values > 1 indicate positive association; < 1 indicate negative association.

conviction $(1 - P(B)) / (1 - \text{confidence})$. Measures departure from independence. Higher = stronger implication.

Value

An object of class "net_association_rules" containing:

rules Data frame with columns: antecedent (list), consequent (list), support, confidence, lift, conviction, count, n_transactions.

frequent_itemsets List of frequent itemsets per level k.

items Character vector of all items.

n_transactions Integer.

n_rules Integer.

params List of min_support, min_confidence, min_lift, max_length.

References

Agrawal, R. & Srikant, R. (1994). Fast algorithms for mining association rules. In *Proc. 20th VLDB Conference*, 487–499.

See Also

[build_network](#), [predict_links](#)

Examples

```
# From a list of transactions
trans <- list(
  c("plan", "discuss", "execute"),
  c("plan", "research", "analyze"),
  c("discuss", "execute", "reflect"),
  c("plan", "discuss", "execute", "reflect"),
  c("research", "analyze", "reflect")
)
rules <- association_rules(trans, min_support = 0.3, min_confidence = 0.5)
print(rules)

# From a netobject (sequences as transactions)
seqs <- data.frame(
  V1 = sample(LETTERS[1:5], 50, TRUE),
  V2 = sample(LETTERS[1:5], 50, TRUE),
  V3 = sample(LETTERS[1:5], 50, TRUE)
)
net <- build_network(seqs, method = "relative")
rules <- association_rules(net, min_support = 0.1)
```

as_htna

Build a grouped node-level network (htna) from data and a clustering

Description

Builds the full node-level network from the original data and attaches a cluster grouping, producing a single netobject in which every actor is a node and cluster membership labels the actors. This is the node-level counterpart of [build_mcml](#): where `build_mcml` collapses the network to a cluster-level (macro) summary, `as_htna` keeps every node and every transition - including the between-cluster transitions an `mcml` only retains in aggregate.

Usage

```
as_htna(x, clusters = NULL, method = "relative", ...)

## S3 method for class 'mcml'
as_htna(x, clusters = NULL, method = "relative", data = NULL, ...)

## Default S3 method:
as_htna(x, clusters = NULL, method = "relative", ...)
```

Arguments

<code>x</code>	Data accepted by <code>build_network</code> (sequence data frame, edgelist, transition matrix, <code>netobject</code> , or <code>tna</code>); or an <code>mcml</code> object, in which case the original data must also be supplied and the <code>mcml</code> provides the cluster membership.
<code>clusters</code>	Cluster assignment: a named list of node-name vectors, a per-node membership vector, or a two-column data frame. When <code>NULL</code> and <code>x</code> carries node groups (or is an <code>mcml</code>), those are used.
<code>method</code>	Estimator passed to <code>build_network</code> . Default "relative" (row-normalized transitions).
<code>...</code>	Further arguments forwarded to <code>build_network</code> (e.g. <code>actor</code> , <code>action</code> , <code>time</code> for long-format data).
<code>data</code>	For the <code>mcml</code> method, the original data the <code>mcml</code> was built from (sequence/edgelist/etc.). Optional when the <code>mcml</code> was built from sequence/edgelist data: <code>build_mcml()</code> stashes that source (and the <code>actor/action/time</code> roles), so <code>as_htna(mcml)</code> works on its own. Required for an <code>mcml</code> built from a matrix/aggregate, which retains no node-level data.

Details

Why this rebuilds from data. An `mcml` stores cluster-level data (the macro sequences are recoded to cluster labels, and the per-cluster data is filtered to within-cluster nodes), so it does not retain a faithful node-level transition network. The only faithful source of node-level between-cluster transitions is the original data. `as_htna()` therefore rebuilds from data via `build_network`; an `mcml` can supply the cluster membership, but the data must be provided.

The result is a genuine `netobject`, so it supports inference (`bootstrap_network`, centrality, `permutation`) and plots directly as a grouped network with `cograph::plot_htna(as_htna(data, clusters))`.

Value

A `netobject` (`cograph_network`) over all nodes, with a `cluster` column on `$nodes`, `$node_groups` populated, and the membership stored in the "cluster_members" attribute.

See Also

`build_mcml`, `build_network`; plot with `cograph::plot_htna()`.

Examples

```
seqs <- data.frame(
  t1 = c("A", "C", "E", "B"), t2 = c("B", "D", "F", "A"),
  t3 = c("C", "A", "E", "D"), stringsAsFactors = FALSE
)
clusters <- list(C1 = c("A", "B"), C2 = c("C", "D"), C3 = c("E", "F"))
net <- as_htna(seqs, clusters)
net$nodes$cluster
## Not run:
cograph::plot_htna(net)
```

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

as_netdifference	<i>Coerce an inferential comparison to a network difference</i>
------------------	---

Description

Coerce an inferential comparison to a network difference

Usage

```
as_netdifference(x, ...)

## S3 method for class 'net_bayes'
as_netdifference(x, significant_only = TRUE, ...)

## S3 method for class 'netdifference'
as_netdifference(x, ...)

## Default S3 method:
as_netdifference(x, ...)
```

Arguments

x	An object with network-difference fields.
...	Additional arguments passed to methods.
significant_only	Logical. For inferential objects, keep only supported differences in the plotted weight matrix while retaining the full difference and interval matrices. Default TRUE.

Value

A netdifference object suitable for `cograph::splot()`.

as_netobject

*Coerce a network object to a Nestimate netobject***Description**

Promotes a **psychnet** result (class `c("psychnet", "cograph_network")`) or any bare `cograph_network` to the dual-class `c("netobject", "cograph_network")` used throughout Nestimate, so it dispatches to the package's verbs (`centrality()`, `plot()`, `bootstrap`, `reliability`, ...). A `netobject` is returned unchanged.

Usage

```
as_netobject(x)

## S3 method for class 'netobject'
as_netobject(x)

## S3 method for class 'psychnet'
as_netobject(x)

## S3 method for class 'cograph_network'
as_netobject(x)

## Default S3 method:
as_netobject(x)
```

Arguments

`x` A `psychnet` object, a `cograph_network`, or a `netobject`.

Details

The `psychnet` method re-derives the integer-indexed edge table that Nestimate expects (`psychnet` stores character-labelled edges), preserves the estimator name in `$method`, and parks every `psychnet`-specific field - including the graphical-lasso `$kkt` optimality certificate - under `$meta$psychnet` so nothing is lost in translation.

Value

A `c("netobject", "cograph_network")` object.

See Also

[validate_netobject](#)

Examples

```
net <- build_cor(data.frame(a = rnorm(50), b = rnorm(50), c = rnorm(50)))
identical(as_netobject(net), net) # netobjects pass through unchanged
```

as_networks

Promote a psychometric MCML result to a network group

Description

as_networks() is the psychometric-network counterpart of [as_tna](#). It promotes the cluster-level (macro) and within-cluster networks produced by [build_mcml_pc](#) into a single netobject_group, so the result flows into the same downstream verbs as any other group of networks (print(), summary(), plot(), [net_centrality](#)).

Usage

```
as_networks(x)

## S3 method for class 'mcml_pc'
as_networks(x)

## Default S3 method:
as_networks(x)
```

Arguments

x An object to convert. The mcml_pc method (from [build_mcml_pc](#)) is the primary path.

Details

Where as_tna() promotes *transition* networks (directed, row-normalised, with initial probabilities) and re-wraps raw matrices, as_networks() promotes *psychometric* networks (undirected; correlation / partial-correlation / glasso). The macro and within-cluster components of an mcml_pc object are already full netobjects carrying their estimator, directedness and data, so this function assembles them into a group rather than re-wrapping matrices.

Value

A netobject_group: a named list whose first element is macro (the cluster-level network), followed by one netobject per non-singleton cluster.

The mcml_pc method returns a netobject_group; singleton clusters (no within-network) are dropped with a warning().

The default method returns the input unchanged if it is already a netobject_group, otherwise it errors.

See Also

[build_mcml_pc](#) to create the input, [as_tna](#) for the transition-network counterpart.

Examples

```
set.seed(1)
df <- as.data.frame(matrix(stats::rnorm(200 * 6), 200, 6))
names(df) <- c("a1", "a2", "a3", "b1", "b2", "b3")
clusters <- list(A = c("a1", "a2", "a3"), B = c("b1", "b2", "b3"))
fit <- build_mcml_pc(df, clusters, aggregation = "composite", method = "cor")
nets <- as_networks(fit)
nets
nets$macro$weights
```

as_tna

Convert cluster_summary to tna Objects

Description

Converts a `cluster_summary` object to proper `tna` objects that can be used with all functions from the `tna` package. Creates both a between-cluster `tna` model (cluster-level transitions) and within-cluster `tna` models (internal transitions within each cluster).

Usage

```
as_tna(x)

## S3 method for class 'mcml'
as_tna(x)

## Default S3 method:
as_tna(x)
```

Arguments

`x` A `cluster_summary` object created by [cluster_summary](#). The aggregated weights are passed to `tna::tna()`, which row-normalises them as needed.

Details

This is the final step in the MCML workflow, enabling full integration with the `tna` package for centrality analysis, bootstrap validation, permutation tests, and visualization.

Requirements:

The `tna` package must be installed. If not available, the function throws an error with installation instructions.

Workflow:


```
# Full MCML workflow
net <- build_network(data, method = "relative")
net$nodes$clusters <- group_assignments
cs <- cluster_summary(net)
tna_models <- as_tna(cs)

# Now use tna package functions
plot(tna_models$macro)
tna::centralities(tna_models$macro)
tna::bootstrap(tna_models$macro, iter = 1000)

# Analyze within-cluster patterns
plot(tna_models$clusters$ClusterA)
tna::centralities(tna_models$clusters$ClusterA)
```

Zero-out-degree (sink) nodes:

Every cluster is returned, regardless of its row sums. A node with zero outgoing weight is a legitimate sink (a terminal state); its row in the wrapped network is left all-zero. This holds for both `net_method = "relative"` and `"frequency"` – the stored weights are never re-normalised, so a sink row needs no special handling. Inspect `rowSums(x$clusters[[cl]]$weights)` to find sink nodes.

Value

A `cluster_tna` object (S3 class) containing:

between A `tna` object representing cluster-level transitions. Contains `$weights` ($k \times k$ transition matrix), `$inits` (initial distribution), and `$labels` (cluster names). Use this for analyzing how learners/entities move between high-level groups or phases.

within Named list of `tna` objects, one per cluster. Each `tna` object represents internal transitions within that cluster. Contains `$weights` ($n_i \times n_i$ matrix), `$inits` (initial distribution), and `$labels` (node labels). Clusters with single nodes or zero-row nodes are excluded (`tna` requires positive row sums).

A `netobject_group` with data preserved from each sub-network.

A `tna` object constructed from the input.

See Also

[cluster_summary](#) to create the input object, `plot()` for visualization without conversion, `tna::tna` for the underlying `tna` constructor

Examples

```
mat <- matrix(runif(36), 6, 6)
rownames(mat) <- colnames(mat) <- LETTERS[1:6]
clusters <- list(G1 = c("A", "B"), G2 = c("C", "D"), G3 = c("E", "F"))
cs <- cluster_summary(mat, clusters)
tna_models <- as_tna(cs)
tna_models
tna_models$macro$weights
```

bayes_compare	<i>Bayesian Dirichlet-Multinomial comparison of two transition networks</i>
---------------	---

Description

Compares two transition networks estimated by [build_network](#) (method "relative" or "frequency") using a Bayesian Dirichlet-Multinomial model. The outgoing transitions from each source state are modelled as a Multinomial draw with a Dirichlet prior on the transition probabilities. With a Jeffreys prior the posterior for the transitions out of state i is $\text{Dirichlet}(c_i + \alpha)$, where c_i are the observed outgoing counts. Each edge probability is then marginally Beta-distributed, so the posterior mean difference between the two networks is available in closed form and a credible interval is obtained by Monte Carlo.

This is a complement to [permutation](#): the permutation test answers "is this difference more extreme than chance?"; the Bayesian comparison answers "what is the plausible range of the true difference, and how precisely is it estimated given the counts?". An edge with few outgoing transitions from its source state yields a wide credible interval even when its row-normalised probability looks decisive.

`bayes_compare()` also accepts two [net_edge_betweenness](#) objects (source method "relative" only). Edge betweenness is a nonlinear function of the whole transition matrix, so instead of Beta marginals the full transition matrix is drawn from each group's row-wise Dirichlet posterior and edge betweenness is recomputed on every draw - the Bayesian analogue of `permutation()`'s edge-betweenness dispatch. The result summarises the posterior of $\text{EB}(x) - \text{EB}(y)$: `diff` is the posterior mean difference, `prob_x/prob_y` hold the posterior mean betweenness matrices, and `observed_diff` the plug-in difference of the two input networks. Both inputs must use the same `invert` setting.

Usage

```
bayes_compare(
  x,
  y = NULL,
  prior = 0.5,
  draws = 10000L,
  ci = 0.95,
  mean_threshold = 0.01,
  bound_threshold = 0.001,
  seed = NULL
)
```

Arguments

- | | |
|---|--|
| x | A <code>netobject</code> (from build_network), a <code>netobject_group</code> , an <code>mcml</code> object, or a net_edge_betweenness object. Must use a transition method ("relative" / "frequency" and their aliases). |
| y | A second object of the same kind as x, or <code>NULL</code> . When x is a <code>netobject_group</code> and y is <code>NULL</code> , all pairwise comparisons among the groups are returned. |

prior	Numeric. Dirichlet prior concentration added to every cell (default 0.5, the Jeffreys prior). Use 1 for a uniform (Laplace) prior.
draws	Integer. Number of Monte Carlo posterior draws used for the credible intervals (default 10000).
ci	Numeric in (0, 1). Credible interval mass (default 0.95).
mean_threshold	Numeric. An edge is flagged significant only if the absolute posterior mean difference exceeds this (default 0.01).
bound_threshold	Numeric. An edge is flagged significant only if the credible-interval bound nearest zero exceeds this in absolute value (default 0.001). Guards against differences that are detectable but negligibly small.
seed	Integer or NULL. RNG seed for reproducible credible intervals.

Value

An object of class `c("net_bayes", "netdifference", "net_permutation")`. It carries the same fields as a [permutation](#) result, so it is a drop-in wherever a `net_permutation` is consumed, and also carries a `netdifference` difference matrix for cograph difference plotting, plus Bayesian extras:

x, y The two input `netobjects`.

diff Posterior mean difference matrix (`prob_x - prob_y`); the analogue of the permutation observed difference.

difference_matrix Alias of `diff` for cograph `netdifference` helpers.

diff_sig Difference where sig, else 0.

p_values P-value matrix (the two-sided Bayesian p-equivalent).

effect_size Posterior mean difference over its posterior SD.

ci_lower, ci_upper Credible-interval bound matrices.

p_difference Probability of the difference: the share of posterior mass on the dominant side of zero, in $[0.5, 1]$ ($P(\text{High} > \text{Low})$ for a positive difference).

p_bayes Alias of `p_values` (two-sided Bayesian p, $2(1 - p_difference)$).

prob_x, prob_y Posterior mean transition-probability matrices.

sig Logical significance matrix (CI excludes zero, mean and nearest bound exceed their thresholds).

summary Long-format data frame whose columns are a superset of `summary.net_permutation` (from, to, weight_x, weight_y, diff, effect_size, p_value, sig) plus `count_x`, `count_y`, `ci_lower`, `ci_upper`, `ci_width`, `p_difference`.

method, iter, alpha, paired, adjust Permutation-compatible settings (`iter = draws`, `alpha = 1 - ci`, `paired = FALSE`, `adjust = "none"`).

prior, draws, ci, mean_threshold, bound_threshold Bayesian settings.

References

- Johnston, L. & Jendoubi, T. (2026). How Delivery Mode Reshapes Resource Engagement: A Bayesian Differential Network Analysis. TNA Workshop 2026.
- Gelman, A., Carlin, J. B., Stern, H. S., Dunson, D. B., Vehtari, A., & Rubin, D. B. (2013). *Bayesian Data Analysis* (3rd ed.). CRC Press.
- Jeffreys, H. (1946). An invariant form for the prior probability in estimation problems. *Proceedings of the Royal Society of London A*, 186(1007), 453-461.

See Also

[permutation](#) for the frequentist complement; [certainty](#) for single-network posterior edge intervals; [subtract_networks](#) and [as_netdifference](#) for the difference verbs; [build_network](#)

Examples

```
s1 <- data.frame(V1 = c("A","B","C"), V2 = c("B","C","A"))
s2 <- data.frame(V1 = c("A","C","B"), V2 = c("C","B","A"))
n1 <- build_network(s1, method = "relative")
n2 <- build_network(s2, method = "relative")
bayes_compare(n1, n2, draws = 500, seed = 1)
```

betti_numbers

Betti Numbers

Description

Computes Betti numbers: β_0 (components), β_1 (loops), β_2 (voids), etc.

Usage

```
betti_numbers(sc)
```

Arguments

sc A simplicial_complex object.

Value

Named integer vector c(b0 = ..., b1 = ..., ...).

Examples

```
mat <- matrix(c(0,.6,.5,.6,0,.4,.5,.4,0), 3, 3)
colnames(mat) <- rownames(mat) <- c("A","B","C")
sc <- build_simplicial(mat, threshold = 0.3)
betti_numbers(sc)
```

bipartite_groups	<i>Hypergraph from bipartite group / event data</i>
------------------	---

Description

Constructs a [net_hypergraph](#) from long-format event data in which each row records a player participating in a group (a session, team, project, transaction, or any group context). Each unique group becomes one hyperedge spanning the players that appeared in it. Optional weight column produces a weighted incidence matrix.

Usage

```
bipartite_groups(data, player, group, weight = NULL)
```

Arguments

data	Data frame in long format. Must contain player and group columns; optionally a weight column.
player	Character. Name of the column whose values become the hypergraph's nodes (players, participants, actors).
group	Character. Name of the column whose values become the hypergraph's hyperedges (groups, sessions, teams).
weight	Character or NULL. If supplied, the column is summed per (player, group) pair to produce a weighted incidence matrix. Default NULL produces a 0/1 binary incidence matrix.

Details

The bipartite representation preserves the full group structure without projecting to a pairwise network. A group of three players A, B, C produces a single 3-hyperedge containing all three, not three pairwise edges AB, AC, BC. This avoids information loss when group interactions are the primary unit of analysis (Perc et al. 2013).

Unlike [build_hypergraph\(\)](#) (which derives hyperedges from a network's clique structure), [bipartite_groups\(\)](#) takes group memberships directly. The two functions are complementary:

- [bipartite_groups\(\)](#) - when group membership is observed (sessions, transactions, co-authorships).
- [build_hypergraph\(\)](#) - when only pairwise interactions are observed and triadic structure must be inferred from triangles.

Rows with NA in either the player or group column (or, when supplied, the weight column) are dropped silently.

Value

A [net_hypergraph](#) object with the same structure produced by [build_hypergraph\(\)](#) (hyperedges, incidence, nodes, n_nodes, n_hyperedges, size_distribution, params). The params list records source = "bipartite_groups" and the original column names.

Note

(experimental) Validated against a hand-computed `table()` incidence reference only; no independent R package exposes the long-format-to-binary-incidence primitive, because the operation is definitionally `table()`. The code path is a direct one-to-one restatement of its definition.

References

Perc, M., Gomez-Gardenes, J., Szolnoki, A., Floria, L. M., & Moreno, Y. (2013). Evolutionary dynamics of group interactions on structured populations: a review. *Journal of the Royal Society Interface* 10(80), 20120997. doi:10.1098/rsif.2012.0997

See Also

`build_hypergraph()` for the clique-based constructor.

Examples

```
df <- data.frame(
  player = c("Alice", "Bob", "Carol", "Alice", "Bob",
            "Dave", "Carol", "Dave", "Eve"),
  session = c("S1", "S1", "S1", "S2", "S2",
            "S3", "S3", "S3", "S3")
)
hg <- bipartite_groups(df, player = "player", group = "session")
print(hg)
summary(hg)
```

bootstrap_network	<i>Bootstrap a Network Estimate</i>
-------------------	-------------------------------------

Description

Non-parametric bootstrap for any network estimated by `build_network`. Works with all built-in methods (transition and association) as well as custom registered estimators.

For transition methods ("relative", "frequency", "co_occurrence"), uses a fast pre-computation strategy: per-sequence count matrices are computed once, and each bootstrap iteration only resamples sequences via `colSums` (C-level) plus lightweight post-processing. Data must be in wide format for transition bootstrap; use `convert_sequence_format` to convert long-format data first.

For association methods ("cor", "pcor", "glasso", and custom estimators), the full estimator is called on resampled rows each iteration.

If a transition network contains only one sequence, the function warns that such a network is not recommended for bootstrap or other confirmatory testing.

Usage

```
bootstrap_network(
  x,
  iter = 1000L,
  ci_level = 0.05,
  inference = "stability",
  consistency_range = c(0.75, 1.25),
  edge_threshold = NULL,
  seed = NULL,
  boundary = c("inclusive", "strict"),
  ci_method = c("percentile", "basic")
)
```

Arguments

x	A netobject from build_network . The data, method, params, scaling, threshold, and level are all extracted from this object.
iter	Integer. Number of bootstrap iterations (default: 1000).
ci_level	Numeric. Significance level for CIs and p-values (default: 0.05).
inference	Character. "stability" (default) tests whether bootstrap replicates fall within a multiplicative consistency range around the original weight. "threshold" tests whether replicates exceed a fixed edge threshold.
consistency_range	Numeric vector of length 2. Multiplicative bounds for stability inference (default: c(0.75, 1.25)).
edge_threshold	Numeric or NULL. Fixed threshold for inference = "threshold". If NULL, defaults to the 10th percentile of absolute original edge weights.
seed	Integer or NULL. RNG seed for reproducibility.
boundary	Character. Comparison rule when computing the consistency-range p-value. "inclusive" (default, tna-compatible) counts iterations that meet the bound ($\leq$ / $\geq$); "strict" counts only iterations strictly outside ($<$ / $>$).
ci_method	Character. Method for the edge-weight confidence intervals. "percentile" (default) uses the empirical bootstrap quantiles (Efron). "basic" reflects those quantiles around the observed weight, $(2\hat{\theta} - q_{1-\alpha/2}, 2\hat{\theta} - q_{\alpha/2})$ (Davison & Hinkley 1997, eq. 5.6), which corrects first-order bootstrap bias but can produce bounds outside the natural weight range near boundaries (e.g., below 0 for transition probabilities close to 0).

Value

An object of class "net_bootstrap" containing:

- original** The original netobject.
- mean** Bootstrap mean weight matrix.
- sd** Bootstrap SD matrix.
- p_values** P-value matrix.

significant Original weights where $p < ci_level$, else 0.
ci_lower Lower CI bound matrix.
ci_upper Upper CI bound matrix.
cr_lower Consistency range lower bound (stability only).
cr_upper Consistency range upper bound (stability only).
summary Long-format data frame of edge-level statistics.
model Pruned netobject (non-significant edges zeroed).
method, params, iter, ci_level, inference Bootstrap config.
consistency_range, edge_threshold Inference parameters.

See Also

[certainty](#) for the closed-form Bayesian counterpart (same result layout, no resampling); [build_network](#), [print.net_bootstrap](#), [summary.net_bootstrap](#)

Examples

```
net <- build_network(data.frame(V1 = c("A", "B", "C"), V2 = c("B", "C", "A")),
  method = "relative")
boot <- bootstrap_network(net, iter = 10)

seqs <- data.frame(
  V1 = sample(LETTERS[1:4], 30, TRUE), V2 = sample(LETTERS[1:4], 30, TRUE),
  V3 = sample(LETTERS[1:4], 30, TRUE), V4 = sample(LETTERS[1:4], 30, TRUE)
)
net <- build_network(seqs, method = "relative")
boot <- bootstrap_network(net, iter = 100)
print(boot)
summary(boot)
```

Description

Fast, single-call bootstrap for EBICglasso partial correlation networks. Combines nonparametric edge/centrality bootstrap, case-dropping stability analysis, edge/centrality difference tests, predictability CIs, and thresholded network into one function. Designed as a faster alternative to boot-net with richer output.

Usage

```
boot_glasso(
  x,
  iter = 1000L,
  cs_iter = 500L,
  cs_drop = seq(0.1, 0.9, by = 0.1),
  alpha = 0.05,
  gamma = 0.5,
  nlambdas = 100L,
  centrality = c("strength", "expected_influence", "betweenness", "closeness"),
  centrality_fn = NULL,
  cor_method = "pearson",
  ncores = 1L,
  seed = NULL
)
```

Arguments

<code>x</code>	A data frame, numeric matrix (observations x variables), or a netobject with <code>method = "glasso"</code> .
<code>iter</code>	Integer. Number of nonparametric bootstrap iterations (default: 1000).
<code>cs_iter</code>	Integer. Number of case-dropping iterations per drop proportion (default: 500).
<code>cs_drop</code>	Numeric vector. Drop proportions for CS-coefficient computation (default: <code>seq(0.1, 0.9, by = 0.1)</code>).
<code>alpha</code>	Numeric. Significance level for CIs (default: 0.05).
<code>gamma</code>	Numeric. EBIC hyperparameter (default: 0.5).
<code>nlambdas</code>	Integer. Number of lambda values in the regularization path (default: 100).
<code>centrality</code>	Character vector. Centrality measures to compute. All four built-in measures ("strength", "expected_influence", "betweenness", "closeness") are computed internally with no extra dependencies and are always taken from the built-in path even if <code>centrality_fn</code> is supplied. Names that are not one of these four are valid only when a <code>centrality_fn</code> is supplied; that function is then responsible for returning them. Default: <code>c("strength", "expected_influence", "betweenness", "closeness")</code> .
<code>centrality_fn</code>	Optional function. A custom centrality function that takes a weight matrix and returns a named list of centrality vectors. When NULL (default), all four built-in measures are computed internally: "strength"/"expected_influence" via <code>rowSums</code> , and "betweenness"/"closeness" via an internal Floyd-Warshall shortest-path routine. When provided, the function is called as <code>centrality_fn(mat)</code> and is used only for requested measures that are not one of the four built-ins; it should return a named list (e.g., <code>list(my_metric = ...)</code>).
<code>cor_method</code>	Character. Correlation method: "pearson" (default), "spearman", or "kendall".
<code>ncores</code>	Integer. Number of parallel cores for <code>mclapply</code> (default: 1, sequential).
<code>seed</code>	Integer or NULL. RNG seed for reproducibility.

Value

An object of class "boot_glasso" containing:

original_pcor Original partial correlation matrix.
original_precision Original precision matrix.
original_centrality Named list of original centrality vectors.
original_predictability Named numeric vector of node R-squared.
edge_ci Data frame of edge CIs (edge, weight, ci_lower, ci_upper, inclusion).
edge_inclusion Named numeric vector of edge inclusion probabilities.
thresholded_pcor Partial correlation matrix with non-significant edges zeroed.
centrality_ci Named list of data frames (node, value, ci_lower, ci_upper) per centrality measure.
cs_coefficient Named numeric vector of CS-coefficients per centrality measure.
cs_data Data frame of case-dropping correlations (drop_prop, measure, correlation).
edge_diff_p Symmetric matrix of pairwise edge difference p-values.
centrality_diff_p Named list of symmetric p-value matrices per centrality measure.
predictability_ci Data frame of node predictability CIs (node, r2, ci_lower, ci_upper).
boot_edges iter x n_edges matrix of bootstrap edge weights.
boot_centrality Named list of iter x p bootstrap centrality matrices.
boot_predictability iter x p matrix of bootstrap R-squared.
nodes Character vector of node names.
n Sample size.
p Number of variables.
iter Number of nonparametric iterations.
cs_iter Number of case-dropping iterations.
cs_drop Drop proportions used.
alpha Significance level.
gamma EBIC hyperparameter.
nlambda Lambda path length.
centrality_measures Character vector of centrality measures.
cor_method Correlation method.
lambda_path Lambda sequence used.
lambda_selected Selected lambda for original data.
timing Named numeric vector with timing in seconds.

See Also

[build_network](#), [bootstrap_network](#)

Examples

```
set.seed(1)
dat <- as.data.frame(matrix(rnorm(60), ncol = 3))
net <- build_network(dat, method = "glasso")
bg <- boot_glasso(net, iter = 10, cs_iter = 5, centrality = "strength")

set.seed(42)
mat <- matrix(rnorm(60), ncol = 4)
colnames(mat) <- LETTERS[1:4]
net <- build_network(as.data.frame(mat), method = "glasso")
boot <- boot_glasso(net, iter = 100, cs_iter = 50, seed = 42,
  centrality = c("strength", "expected_influence"))
print(boot)
summary(boot, type = "edges")
```

bottleneck_distance *Bottleneck Distance Between Persistence Diagrams*

Description

Computes the bottleneck distance between two persistence diagrams. For finite pairs, the bottleneck distance is

$$W_{\infty}(D_1, D_2) = \inf_{\gamma} \sup_{p \in D_1} \|p - \gamma(p)\|_{\infty},$$

where γ ranges over bijections $D_1 \cup \Delta \rightarrow D_2 \cup \Delta$ and $\Delta = \{(x, x)\}$ is the diagonal. Each point may match a point in the other diagram or its projection onto the diagonal at cost $|d - b|/2$. Computed via binary search on ε plus a Kuhn bipartite-matching feasibility check.

Essential classes (death = Inf in VR mode, or death = 0 in clique mode) are matched one-to-one within each dimension. If the diagrams have different numbers of essential classes in some dimension, the bottleneck distance for that dimension is Inf.

Usage

```
bottleneck_distance(d1, d2, dimension = NULL, tol = .Machine$double.eps^0.5)
```

Arguments

d1, d2	persistent_homology objects, or data.frames with columns dimension, birth, death.
dimension	Integer vector of dimensions to compare. NULL (default) compares all dimensions appearing in either diagram and returns a named numeric vector.
tol	Numerical tolerance for binary search (default <code>.Machine\$double.eps ^ 0.5</code>).

Value

Named numeric vector. Names are "dim_<k>". Inf indicates a structural mismatch (different essential counts in that dimension); a self-distance is always 0.

References

Edelsbrunner, H. & Harer, J. (2010). *Computational Topology: An Introduction*. AMS. Section VIII.

Examples

```
mat1 <- matrix(c(0, .6, .5, .6, 0, .4, .5, .4, 0), 3, 3)
rownames(mat1) <- colnames(mat1) <- c("A", "B", "C")
ph1 <- persistent_homology(mat1, n_steps = 5)
bottleneck_distance(ph1, ph1) # self-distance is 0
```

 build_atna

Build an Attention-Weighted Transition Network (ATNA)

Description

Convenience wrapper for `build_network(method = "attention")`. Computes decay-weighted transitions from sequence data.

Usage

```
build_atna(data, start = FALSE, end = FALSE, ...)
```

Arguments

data	Data frame (sequences or per-observation frequencies) or a square symmetric matrix (correlation or covariance).
start	Boundary marker prepended to every sequence as an explicit start state (a pure source: no incoming edges, every sequence's first transition is start -> first_observed). FALSE (default) adds nothing; TRUE uses the label "Start"; a single string uses that string as the label. Only valid for the transition methods (relative, frequency, co_occurrence, attention); errors otherwise.
end	Boundary marker placed in the single cell after each sequence's last observed (non-NA) state, as an explicit terminal state (a pure sink: no outgoing edges, no self-loop – distinct from mark_terminal_state , which fills all trailing NAs into an absorbing state). FALSE (default) adds nothing; TRUE uses the label "End"; a single string uses that string as the label. Same method restriction as start.
...	Additional arguments passed to build_network .

Value

A netobject (see [build_network](#)).

See Also

[build_network](#)

Examples

```
seqs <- data.frame(V1 = c("A", "B", "C"), V2 = c("B", "C", "A"))
net <- build_atna(seqs)
```

build_clusters

Cluster Sequences by Dissimilarity

Description

Clusters wide-format sequences using pairwise string dissimilarity and either PAM (Partitioning Around Medoids) or hierarchical clustering. Supports 9 distance metrics including temporal weighting for Hamming distance. When the **stringdist** package is available, uses C-level distance computation for 100-1000x speedup on edit distances.

Usage

```
build_clusters(
  data,
  k,
  dissimilarity = "hamming",
  method = "pam",
  na_syms = c("*", "%"),
  weighted = FALSE,
  lambda = 1,
  seed = NULL,
  q = 2L,
  p = 0.1,
  covariates = NULL,
  estimator = c("auto", "firth", "multinom", "chisq"),
  ...
)
```

Arguments

data Input data. Accepts multiple formats:
data.frame / matrix Wide-format sequences (rows = sequences, columns = time points, values = state names).

	netobject A network object from <code>build_network</code>. Extracts the stored sequence data. Only valid for sequence-based methods (relative, frequency, co_occurrence, attention). tna A tna model from the tna package. Decodes the integer-encoded sequence data using stored labels. cograph_network A cograph network object. Extracts the stored sequence data.
k	Integer. Number of clusters (must be between 2 and <code>nrow(data) - 1</code>).
dissimilarity	Character. Distance metric. One of "hamming", "osa" (optimal string alignment), "lv" (Levenshtein), "dl" (Damerau-Levenshtein), "lcs" (longest common subsequence), "qgram", "cosine", "jaccard", "jw" (Jaro-Winkler). Default: "hamming".
method	Character. Clustering method. "pam" for Partitioning Around Medoids, or a hierarchical method: "ward.D2", "ward.D", "complete", "average", "single", "mcquitty", "median", "centroid". Default: "pam".
na_syms	Character vector. Symbols treated as missing values. Default: <code>c("*", "%")</code>. Missing-value distance rule: after symbols are converted to NA, missing values are encoded as a single comparable sentinel state – <i>not</i> pairwise-deleted. Two missing values in the same position match (distance contribution 0); a missing value paired with any observed state mismatches (distance contribution 1 for Hamming, etc.). This is the conventional behaviour for aligned sequence matrices because pairwise deletion would change the effective length of every pair and break the metric. If you want pairwise deletion or a different missing-value semantic, drop or recode the missing cells before passing the data in.
weighted	Logical. Apply exponential decay weighting to Hamming distance positions? Only valid when <code>dissimilarity = "hamming"</code> . Default: FALSE.
lambda	Numeric. Non-negative decay rate for weighted Hamming. Higher values weight earlier positions more strongly. Default: 1.
seed	Integer or NULL. Random seed for reproducibility. Default: NULL.
q	Integer. Size of q-grams for "qgram", "cosine", and "jaccard" distances. Default: 2L.
p	Numeric. Winkler prefix penalty for Jaro-Winkler distance. Must be between 0 and 0.25. Default: 0.1.
covariates	Optional. Post-hoc covariate analysis of cluster membership. Accepts: string Single column name, e.g. "Age". Resolved against <code>x\$metadata</code> (and <code>x\$data</code>) for <code>netobject</code> or <code>cograph_network</code> input. character vector <code>c("Age", "Gender")</code>, same lookup. formula <code>~ Age + Gender</code>, same lookup; supports "Age + Gender" string form too. data.frame All columns used as covariates verbatim; must have one row per sequence. NULL No covariate analysis (default). For <code>netobject</code> or <code>cograph_network</code> input, names are resolved against <code>\$metadata</code> first and then non-state columns of <code>\$data</code>, so a typical call looks like <code>build_clusters(net,</code>

`k = 3, covariates = "session_label")` without pre-extracting a `data.frame`. tna input requires the `data.frame` form. Results are stored in `$covariates`.

estimator Multinomial logit fitter for the covariate analysis. "auto" (default) inspects the cluster x covariate cross-tab and falls back to "firth" only when any cell has fewer than 5 observations (quasi-complete separation risk); otherwise uses the much faster "multinom". "firth" forces Firth's penalised likelihood via `brglm2::brmultinom` – bias-reduced and finite under separation, but ~200x slower than `multinom` on well-conditioned data. "multinom" forces classical ML via `nnet::multinom`; warns because rare-cell separation produces astronomical ORs with degenerate CIs (silent failure). "chisq" runs WeightedCluster-style descriptive tests (chi-square + Cramer's V + standardized adjusted residuals for factors; Kruskal-Wallis + eta-squared for numerics).

... Unsupported. Supplying unused arguments raises an error.

Value

An object of class "net_clustering" containing:

data The original input data.

k Number of clusters.

assignments Named integer vector of cluster assignments.

silhouette Overall average silhouette width.

sizes Named integer vector of cluster sizes.

method Clustering method used.

dissimilarity Distance metric used.

distance The computed dissimilarity matrix (dist object).

medoids Integer vector of medoid row indices (PAM only; NULL for hierarchical methods).

seed Seed used (or NULL).

weighted Logical, whether weighted Hamming was used.

lambda Lambda value used (0 if not weighted).

Examples

```
seqs <- data.frame(V1 = c("A", "B", "C", "A", "B"), V2 = c("B", "C", "A", "B", "A"),
                  V3 = c("C", "A", "B", "C", "B"))
cl <- build_clusters(seqs, k = 2)
cl

seqs <- data.frame(
  V1 = sample(LETTERS[1:3], 20, TRUE), V2 = sample(LETTERS[1:3], 20, TRUE),
  V3 = sample(LETTERS[1:3], 20, TRUE), V4 = sample(LETTERS[1:3], 20, TRUE)
)
cl <- build_clusters(seqs, k = 2)
print(cl)
summary(cl)
```

build_cna	<i>Build a Co-occurrence Network (CNA)</i>
-----------	--

Description

Convenience wrapper for `build_network(method = "co_occurrence")`. Computes co-occurrence counts from binary or sequence data.

Usage

```
build_cna(data, start = FALSE, end = FALSE, ...)
```

Arguments

data	Data frame (sequences or per-observation frequencies) or a square symmetric matrix (correlation or covariance).
start	Boundary marker prepended to every sequence as an explicit start state (a pure source: no incoming edges, every sequence's first transition is start -> first_observed). FALSE (default) adds nothing; TRUE uses the label "Start"; a single string uses that string as the label. Only valid for the transition methods (relative, frequency, co_occurrence, attention); errors otherwise.
end	Boundary marker placed in the single cell after each sequence's last observed (non-NA) state, as an explicit terminal state (a pure sink: no outgoing edges, no self-loop – distinct from <code>mark_terminal_state</code> , which fills all trailing NAs into an absorbing state). FALSE (default) adds nothing; TRUE uses the label "End"; a single string uses that string as the label. Same method restriction as start.
...	Additional arguments passed to <code>build_network</code> .

Value

A netobject (see `build_network`).

See Also

`build_network`, `cooccurrence` for delimited-field, bipartite, and other non-sequence co-occurrence formats.

Examples

```
seqs <- data.frame(V1 = c("A", "B", "C"), V2 = c("B", "C", "A"))
net <- build_cna(seqs)
```

`build_cor`*Build a Correlation Network*

Description

Convenience wrapper for `build_network(method = "cor")`. Computes Pearson correlations from numeric data.

Usage

```
build_cor(data, ...)
```

Arguments

<code>data</code>	Data frame (sequences or per-observation frequencies) or a square symmetric matrix (correlation or covariance).
<code>...</code>	Additional arguments passed to build_network .

Value

A netobject (see [build_network](#)).

See Also

[build_network](#)

Examples

```
data(srl_strategies)
net <- build_cor(srl_strategies)
```

`build_ftna`*Build a Frequency Transition Network (FTNA)*

Description

Convenience wrapper for `build_network(method = "frequency")`. Computes raw transition counts from sequence data.

Usage

```
build_ftna(data, start = FALSE, end = FALSE, ...)
```

Arguments

<code>data</code>	Data frame (sequences or per-observation frequencies) or a square symmetric matrix (correlation or covariance).
<code>start</code>	Boundary marker prepended to every sequence as an explicit start state (a pure source: no incoming edges, every sequence's first transition is start -> first_observed). FALSE (default) adds nothing; TRUE uses the label "Start"; a single string uses that string as the label. Only valid for the transition methods (relative, frequency, co_occurrence, attention); errors otherwise.
<code>end</code>	Boundary marker placed in the single cell after each sequence's last observed (non-NA) state, as an explicit terminal state (a pure sink: no outgoing edges, no self-loop – distinct from <code>mark_terminal_state</code> , which fills all trailing NAs into an absorbing state). FALSE (default) adds nothing; TRUE uses the label "End"; a single string uses that string as the label. Same method restriction as start.
<code>...</code>	Additional arguments passed to <code>build_network</code> .

Value

A netobject (see `build_network`).

See Also

`build_network`

Examples

```
seqs <- data.frame(V1 = c("A", "B", "C"), V2 = c("B", "C", "A"))
net <- build_ftna(seqs)
```

build_gimme

GIMME: Group Iterative Multiple Model Estimation

Description

Estimates person-specific directed networks from intensive longitudinal data using the unified Structural Equation Modeling (uSEM) framework. Implements a data-driven search that identifies:

1. **Group-level paths:** Directed edges present for a majority (default 75\
2. **Individual-level paths:** Additional edges specific to each person, found after group paths are established.

Uses lavaan for SEM estimation and modification indices. Accepts a single data frame with an ID column (not CSV directories).

Usage

```

build_gimme(
  data,
  vars,
  id,
  time = NULL,
  ar = TRUE,
  standardize = FALSE,
  groupcutoff = 0.75,
  subcutoff = 0.5,
  paths = NULL,
  exogenous = NULL,
  hybrid = FALSE,
  rmsea_cutoff = 0.05,
  srmr_cutoff = 0.05,
  nnfi_cutoff = 0.95,
  cfi_cutoff = 0.95,
  n_excellent = 2L,
  seed = NULL
)

```

Arguments

<code>data</code>	A data.frame in long format with columns for person ID, time-varying variables, and optionally a time/beep column.
<code>vars</code>	Character vector of variable names to model.
<code>id</code>	Character string naming the person-ID column.
<code>time</code>	Character string naming the time/order column, or NULL. When provided, data is sorted by id then time before lagging.
<code>ar</code>	Logical. If TRUE (default), autoregressive paths (each variable predicting itself at lag 1) are included as fixed paths.
<code>standardize</code>	Logical. If TRUE (default FALSE), variables are standardized per person before estimation. Note: the returned coefficient network (<code>\$coefs</code> , <code>\$psi</code> , <code>\$temporal_avg</code> , <code>\$contemporaneous_avg</code> , <code>\$group_paths</code>) is unaffected because Nestimate extracts the standardized lavaan solution (<code>lavInspect(fit, "std")</code>), which is invariant to input scaling. Only the scale-dependent <code>\$fit</code> statistics (<code>chisq</code> , <code>aic</code> , <code>bic</code>) change.
<code>groupcutoff</code>	Numeric between 0 and 1. Proportion of individuals for whom a path must be significant to be added at group level. Default 0.75.
<code>subcutoff</code>	Numeric. Not used (reserved for future subgrouping).
<code>paths</code>	Character vector of lavaan-syntax paths to force into the model (e.g., "V2~V1lag"). Default NULL.
<code>exogenous</code>	Character vector of variable names to treat as exogenous. Default NULL.
<code>hybrid</code>	Logical. If TRUE, also searches residual covariances. Default FALSE.
<code>rmsea_cutoff</code>	Numeric. RMSEA threshold for excellent fit (default 0.05).

srmr_cutoff	Numeric. SRMR threshold for excellent fit (default 0.05).
nnfi_cutoff	Numeric. NNFI/TLI threshold for excellent fit (default 0.95).
cfi_cutoff	Numeric. CFI threshold for excellent fit (default 0.95).
n_excellent	Integer. Number of fit indices that must be excellent to stop individual search. Default 2.
seed	Integer or NULL. Random seed for reproducibility.

Value

An S3 object of class "net_gimme" containing:

temporal $p \times p$ matrix of group-level temporal (lagged) path counts – entry $[i, j]$ = number of individuals with path $j(t-1) \rightarrow i(t)$.

contemporaneous $p \times p$ matrix of group-level contemporaneous path counts – entry $[i, j]$ = number of individuals with path $j(t) \rightarrow i(t)$.

coefs List of per-person $p \times 2p$ coefficient matrices (rows = endogenous, cols = [lagged, contemporaneous]).

psi List of per-person residual covariance matrices.

fit Data frame of per-person fit indices (chisq, df, pvalue, rmsea, srmr, nnfi, cfi, bic, aic, logl, status).

path_counts $p \times 2p$ matrix: how many individuals have each path.

paths List of per-person character vectors of lavaan path syntax.

group_paths Character vector of group-level paths found.

individual_paths List of per-person character vectors of individual-level paths (beyond group).

syntax List of per-person full lavaan syntax strings.

labels Character vector of variable names.

n_subjects Integer. Number of individuals.

n_obs Integer vector. Time points per individual.

config List of configuration parameters.

See Also

[build_network](#)

Examples

```
# Create simple panel data (3 subjects, 4 variables, 50 time points).
set.seed(42)
n_sub <- 3; n_t <- 50; vars <- paste0("V", 1:4)
rows <- lapply(seq_len(n_sub), function(i) {
  d <- as.data.frame(matrix(rnorm(n_t * 4), ncol = 4))
  names(d) <- vars; d$id <- i; d
})
panel <- do.call(rbind, rows)
res <- build_gimme(panel, vars = vars, id = "id")
print(res)
```

build_glasso	<i>Build a Graphical Lasso Network (EBICglasso)</i>
--------------	---

Description

Convenience wrapper for `build_network(method = "glasso")`. Computes L1-regularized partial correlations with EBIC model selection.

Usage

```
build_glasso(data, ...)
```

Arguments

data	Data frame (sequences or per-observation frequencies) or a square symmetric matrix (correlation or covariance).
...	Additional arguments passed to build_network .

Value

A netobject (see [build_network](#)).

See Also

[build_network](#)

Examples

```
data(srl_strategies)
net <- build_glasso(srl_strategies)
```

build_hon	<i>Build a Higher-Order Network (HON)</i>
-----------	---

Description

Constructs a Higher-Order Network from sequential data, faithfully implementing the BuildHON algorithm (Xu, Wickramaratne & Chawla, 2016).

The algorithm detects when a first-order Markov model is insufficient to capture sequential dependencies and automatically creates higher-order nodes. Uses KL-divergence to determine whether extending a node's history provides significantly different transition distributions.

Usage

```
build_hon(
  data,
  max_order = 5L,
  min_freq = 1L,
  collapse_repeats = FALSE,
  method = "hon+"
)
```

Arguments

data	One of: • <code>data.frame</code>: rows are trajectories, columns are time steps. Trailing NAs are stripped. All non-NA values are coerced to character. • <code>list</code>: each element is a character (or coercible) vector representing one trajectory. • <code>tna</code>: a <code>tna</code> object with sequence data. Numeric state IDs are automatically converted to label names. • <code>netobject</code>: a <code>netobject</code> with sequence data.
max_order	Integer. Maximum order of the HON. Default 5. The algorithm may produce lower-order nodes if the data do not justify higher orders.
min_freq	Integer. Minimum frequency for a transition to be considered. Transitions observed fewer than <code>min_freq</code> times are treated as zero. Default 1.
collapse_repeats	Logical. If TRUE, adjacent duplicate states within each trajectory are collapsed before analysis. Default FALSE.
method	Character. Algorithm to use: "hon+" (default, parameter-free BuildHON+ with lazy observation building and MaxDivergence pruning) or "hon" (original BuildHON with eager observation building).

Details

Node naming convention: Higher-order nodes use readable arrow notation. A first-order node is simply "A". A second-order node representing the context "came from A, now at B" is "A -> B". Third-order: "A -> B -> C", etc.

Algorithm overview:

1. Count all subsequence transitions up to `max_order + 1`.
2. Build probability distributions, filtering by `min_freq`.
3. For each first-order source, recursively test whether extending the history (adding more context) produces a significantly different distribution (via KL-divergence vs. an adaptive threshold).
4. Build the network from the accepted rules, rewiring edges so higher-order nodes are properly connected.

Value

An S3 object of class "net_hon" containing:

matrix Weighted adjacency matrix (rows = from, cols = to). Rows and columns use readable arrow notation (e.g., "A -> B").

edges Data frame with columns: path (full state sequence, e.g., "A -> B -> C"), from (context/conditioning states), to (predicted next state), count (raw frequency), probability (transition probability), from_order, to_order.

nodes data.frame with columns id, label, name (one row per HON node; label/name are the arrow-notation node names). Stored as a data.frame for cograph_network compatibility.

n_nodes Number of HON nodes.

n_edges Number of edges.

first_order_states Character vector of unique original states.

max_order_requested The max_order parameter used.

max_order_observed Highest order actually present.

min_freq The min_freq parameter used.

n_trajectories Number of trajectories after parsing.

directed Logical. Always TRUE.

References

Xu, J., Wickramaratne, T. L., & Chawla, N. V. (2016). Representing higher-order dependencies in networks. *Science Advances*, 2(5), e1600028.

Saebi, M., Xu, J., Kaplan, L. M., Ribeiro, B., & Chawla, N. V. (2020). Efficient modeling of higher-order dependencies in networks: from algorithm to application for anomaly detection. *EPJ Data Science*, 9(1), 15.

Examples

```
seqs <- list(c("A","B","C","D"), c("A","B","C","A"), c("B","C","D","A"))
hon <- build_hon(seqs, max_order = 2)
```

```
# From list of trajectories
trajs <- list(
  c("A", "B", "C", "D", "A"),
  c("A", "B", "D", "C", "A"),
  c("A", "B", "C", "D", "A")
)
hon <- build_hon(trajs, max_order = 3, min_freq = 1)
print(hon)
summary(hon)
```

```
# From data.frame (rows = trajectories)
df <- data.frame(T1 = c("A", "A"), T2 = c("B", "B"),
                 T3 = c("C", "D"), T4 = c("D", "C"))
hon <- build_hon(df, max_order = 2)
```

 build_honem

Build HONEM Embeddings for Higher-Order Networks

Description

Constructs low-dimensional embeddings from a Higher-Order Network (HON) that preserve higher-order dependencies. Uses exponentially-decaying matrix powers of the HON transition matrix followed by truncated SVD.

Usage

```
build_honem(hon, dim = 32L, max_power = 10L)
```

Arguments

hon	A <code>net_hon</code> object from build_hon , or a square weighted adjacency matrix.
dim	Integer. Embedding dimension (default 32).
max_power	Integer. Maximum walk length for neighborhood computation (default 10). Higher values capture longer-range structure.

Details

HONEM is parameter-free and scalable - no random walks, skip-gram, or hyperparameter tuning required.

Value

An object of class `net_honem` with components:

embeddings Numeric matrix (`n_nodes` x `dim`) of node embeddings.

nodes Character vector of node names.

singular_values Numeric vector of top singular values.

explained_variance Proportion of variance explained.

dim Embedding dimension used.

max_power Maximum power used.

n_nodes Number of nodes embedded.

References

Saebi, M., Ciampaglia, G. L., Kazemzadeh, S., & Meyur, R. (2020). HONEM: Learning Embedding for Higher Order Networks. *Big Data*, 8(4), 255-269.

Examples

```
seqs <- list(c("A","B","C","D"), c("A","B","C","A"), c("B","C","D","A"))
hem <- build_honem(build_hon(seqs, max_order = 2), dim = 2)

trajs <- list(c("A","B","C","D"), c("A","B","D","C"),
             c("B","C","D","A"), c("C","D","A","B"))
hon <- build_hon(trajs, max_order = 2)
emb <- build_honem(hon, dim = 4)
print(emb)
plot(emb)
```

build_hypa

*Detect Path Anomalies via HYPA***Description**

Constructs a k -th order De Bruijn graph from sequential trajectory data and uses a hypergeometric null model to detect paths with anomalous frequencies. Paths occurring more or less often than expected under the null model are flagged as over- or under-represented.

Usage

```
build_hypa(
  data,
  order = 2L,
  alpha = 0.05,
  min_count = 5L,
  p_adjust = "BH",
  k = NULL
)
```

Arguments

data	A data.frame (rows = trajectories), list of character vectors, tna object, or netobject with sequence data. For tna/netobject, numeric state IDs are automatically converted to label names.
order	Integer scalar or integer vector. Order(s) of the De Bruijn graph (default 2L). An order of k detects anomalies in paths of length k . When a vector is supplied, one De Bruijn layer is built per order and the per-order results are stored in <code>\$by_order</code> (named by order). The orders are sorted ascending internally, so the <code>cograph_network</code> slots (<code>\$weights</code> , <code>\$edges</code> , <code>\$adjacency</code> , <code>\$xi</code> , <code>\$nodes</code> , <code>\$meta</code>) always describe the network of the <i>lowest order that produced a layer</i> (a requested order with no edges is dropped from <code>\$by_order</code> , <code>\$order</code> and <code>\$k</code>), regardless of the order in which the vector is given; <code>\$scores</code> aggregates every built order.

<code>alpha</code>	Numeric. Significance threshold for anomaly classification (default 0.05). Paths with HYPA score < alpha are under-represented; paths with score > 1-alpha are over-represented.
<code>min_count</code>	Integer. Minimum observed count for a path to be classified as anomalous (default 5). Paths with fewer observations are always classified as "normal" regardless of their HYPA score, since rare occurrences are unreliable.
<code>p_adjust</code>	Character. Method for multiple testing correction of p-values. Default "BH" (Benjamini-Hochberg FDR control). Accepts any method from p.adjust.methods or "none" to skip correction. Under- and over-representation p-values are adjusted separately (two-sided testing).
<code>k</code>	Deprecated. Former name of order; if supplied it overrides order and emits a deprecation message. Use order instead.

Value

An object of class `c("net_hypa", "cograph_network")` with components:

scores Data frame with path, from, to, observed, expected, ratio, p_value, p_under, p_over, p_adjusted_under, p_adjusted_over, anomaly, order columns (one block of rows per requested order). The path column shows the full state sequence (e.g., "A -> B -> C"); from is the context (conditioning states); to is the next state; ratio is observed / expected; p_value is retained as an alias for p_under, the raw lower-tail hypergeometric CDF value; p_over is the inclusive upper-tail probability $P(X \geq \text{observed})$; p_adjusted_under and p_adjusted_over are the corrected p-values for under- and over-representation tests respectively.

ho_edges Alias for scores (all orders, arrow notation).

over Subset of scores classified as over-represented.

under Subset of scores classified as under-represented.

adjacency Weighted adjacency matrix of the lowest-order De Bruijn graph.

weights cograph weight matrix of the lowest-order graph.

xi Fitted propensity matrix of the lowest-order graph.

edges cograph edge data.frame of the lowest-order graph.

by_order Named list of per-order result lists.

order Integer vector of orders actually built (sorted ascending).

k Back-compatibility alias for order.

alpha Significance threshold used.

p_adjust Multiple testing correction method used.

n_anomalous Number of anomalous paths detected (all orders).

n_over Number of over-represented paths (all orders).

n_under Number of under-represented paths (all orders).

n_edges Total number of edges (all orders).

nodes data.frame (id, label, name) of the lowest-order De Bruijn graph nodes (arrow notation).

directed Logical. Always TRUE.

meta cograph meta list of the lowest-order graph.

node_groups Always NULL.

References

LaRock, T., Nanumyan, V., Scholtes, I., Casiraghi, G., Eliassi-Rad, T., & Schweitzer, F. (2020). HYPA: Efficient Detection of Path Anomalies in Time Series Data on Networks. *SDM 2020*, 460-468.

Examples

```
seqs <- list(c("A","B","C"), c("B","C","A"), c("A","C","B"), c("A","B","C"))
hyp <- build_hypa(seqs, order = 2)
```

```
trajs <- list(c("A","B","C"), c("A","B","C"), c("A","B","C"),
             c("A","B","D"), c("C","B","D"), c("C","B","A"))
h <- build_hypa(trajs, order = 2)
print(h)
```

build_hypergraph	<i>Higher-order hypergraph from a network's clique structure</i>
------------------	--

Description

Takes a network and produces a hypergraph by promoting k -cliques ($k \geq 3$) to k -hyperedges. Each k -clique is independently included as a k -hyperedge with probability p . Optionally retains the underlying pairwise edges as 2-hyperedges. Foundation for higher-order analyses.

Usage

```
build_hypergraph(
  net,
  p = 1,
  method = c("clique", "vr", "rips"),
  include_pairwise = TRUE,
  max_size = 3L,
  threshold = 0,
  seed = NULL
)

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

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

Arguments

net	A netobject, cograph_network, simplicial_complex, or numeric adjacency / weight matrix. Directed inputs are symmetrised by the underlying clique enumerator.
p	Probability in $[0, 1]$ that each k -clique with $k \geq 3$ becomes a k -hyperedge. Default 1 (deterministic - every found clique is included).
method	Hyperedge enumeration. "clique" (default) promotes k -cliques in the binarised adjacency to k -hyperedges. A metric Vietoris-Rips construction is not implemented ; "vr" / "rips" are accepted by <code>match.arg</code> but raise an error rather than silently aliasing "clique".
include_pairwise	Logical. Include 2-edges from the input network as 2-hyperedges. Default TRUE. Set FALSE for a "fully higher-order" hypergraph containing only k -hyperedges with $k \geq 3$.
max_size	Integer ≥ 2 . Maximum hyperedge size to extract. Default 3L (triangles only). 4L also includes 4-cliques as 4-hyperedges, etc.
threshold	Numeric. Edge weight cutoff used to binarise the adjacency for clique enumeration. Default 0 (any non-zero weight is an edge).
seed	Optional integer for reproducible Bernoulli sampling when $0 < p < 1$.
x	A net_hypergraph object (for print).
...	Additional arguments (ignored).
object	A net_hypergraph object (for summary).

Details

The construction follows Burgio, Matamalas, Gomez & Arenas (2020) on simplicial / hypergraph contagion. For each k -clique with $k \geq 3$ found in the underlying graph (via `build_simplicial()`), an independent Bernoulli(p) trial decides whether that clique becomes a k -hyperedge. Underlying pairwise edges are always retained when `include_pairwise = TRUE`, so the resulting hypergraph contains both the original 2-edge structure and the sampled higher-order interactions.

At the limits:

- $p = 0$ with `include_pairwise = TRUE` reproduces the input pairwise network as a hypergraph of size-2 edges.
- $p = 1$ with `include_pairwise = FALSE` returns a fully higher-order hypergraph containing only the k -hyperedges ($k \geq 3$) found in the network's clique complex.

Value

A net_hypergraph object: a list with components

hyperedges List of integer vectors. Each entry is a hyperedge given as the sorted node indices it spans.

incidence Numeric matrix of size `n_nodes` x `n_hyperedges`. `incidence[i, j] = 1` iff node i belongs to hyperedge j . Row names are node names; column names are `h1, h2, ...`

nodes Character vector of node names.
 n_nodes, n_hyperedges Scalar counts.
 size_distribution Named integer vector: number of hyperedges of each size, named size_2, size_3, ...
 params Recorded call parameters: method, p, include_pairwise, max_size, threshold, seed.
 The input x invisibly.
 The input object invisibly.

References

Burgio, G., Matamalas, J. T., Gomez, S., & Arenas, A. (2020). Evolution of cooperation in the presence of higher-order interactions: from networks to hypergraphs. *Entropy* 22(7), 744. doi:10.3390/e22070744

See Also

[build_simplicial\(\)](#) (underlying clique enumeration), [build_network\(\)](#).

Examples

```
set.seed(1)
n <- 8
adj <- matrix(stats::rbinom(n * n, 1, 0.5), n, n)
diag(adj) <- 0
adj <- (adj + t(adj)) > 0
rownames(adj) <- colnames(adj) <- LETTERS[seq_len(n)]
hg <- build_hypergraph(adj, p = 1, max_size = 3L)
print(hg)
summary(hg)
```

build_ising

Build an Ising Network

Description

Convenience wrapper for `build_network(method = "ising")`. Computes L1-regularized logistic regression network for binary data.

Usage

```
build_ising(data, ...)
```

Arguments

data Data frame (sequences or per-observation frequencies) or a square symmetric matrix (correlation or covariance).
 ... Additional arguments passed to [build_network](#).

Value

A netobject (see [build_network](#)).

See Also

[build_network](#)

Examples

```
if (requireNamespace("glmnet", quietly = TRUE)) {
  bin_data <- data.frame(matrix(rbinom(200, 1, 0.5), ncol = 5))
  net <- build_ising(bin_data)
}
```

build_mcml

Build MCML from Raw Transition Data

Description

Builds a Multi-Cluster Multi-Level (MCML) model from raw transition data (edge lists or sequences) by recoding node labels to cluster labels and counting actual transitions. Unlike [cluster_summary](#) which aggregates a pre-computed weight matrix, this function works from the original transition data to produce the TRUE Markov chain over cluster states.

Usage

```
build_mcml(
  x,
  clusters = NULL,
  method = c("sum", "mean", "median", "max", "min", "density", "geomean"),
  type = c("tna", "frequency", "cooccurrence", "raw"),
  directed = TRUE,
  compute_within = TRUE,
  actor = NULL,
  action = NULL,
  time = NULL,
  order = NULL,
  session = NULL,
  time_threshold = 900,
  labels = NULL
)
```

Arguments

x	Input data. Accepts multiple formats: data.frame with from/to columns Edge list. Columns named from/source/src/v1/node1/i and to/target/tgt/v2/node2/j are auto-detected. Optional weight column (weight/w/value/strength). data.frame without from/to columns Sequence data. Each row is a sequence, columns are time steps. Consecutive pairs (t, t+1) become transitions. tna object If x\$data is non-NULL, uses sequence path on the raw data. Otherwise falls back to cluster_summary. netobject If x\$data is non-NULL, detects edge list vs sequence data. Otherwise falls back to cluster_summary. cluster_summary Returns as-is. square numeric matrix Falls back to cluster_summary. non-square or character matrix Treated as sequence data.
clusters	Cluster/group assignments. Accepts: named list Direct mapping. List names = cluster names, values = character vectors of node labels. Example: <code>list(A = c("N1", "N2"), B = c("N3", "N4"))</code> data.frame A data frame where the first column contains node names and the second column contains group/cluster names. Example: <code>data.frame(node = c("N1", "N2", "N3"), group = c("A", "A", "B"))</code> membership vector Character or numeric vector. Node names are extracted from the data. Example: <code>c("A", "A", "B", "B")</code> column name string For edge list data.frames, the name of a column containing cluster labels. The mapping is built from unique (node, group) pairs in both from and to columns. Limitation: this mode assigns the row's group label to <i>both</i> endpoints, so it only makes sense for edge lists where source and target nodes always share the same group (within-group edges only). For general edge lists where a single node may be source in some rows and target in others, or where source and target belong to different groups, pass an explicit named list (<code>list(G1 = c("N1", "N2"), ...)</code>) or a two-column data frame <code>data.frame(node, group)</code> instead. NULL Auto-detect from <code>netobject\$nodes</code> or <code>\$node_groups</code> (same logic as cluster_summary).
method	Aggregation method for combining edge weights: "sum", "mean", "median", "max", "min", "density", "geomean". Default "sum". For raw sequence/event-log inputs the function is counting observed transitions, so "sum" is the only interpretation that preserves the count semantics – the other methods are useful when aggregating weighted edge lists or pre-existing weight matrices, where each row already represents a measurement rather than a single observation.
type	Post-processing of the aggregated count matrix. One of: "tna" (default) Row-normalize so each row sums to 1 (first-order Markov transition probabilities). "raw" Return the un-normalized count matrix. "frequency" Explicit alias of "raw" – identical raw count construction (kept as a synonym for callers using frequency-network terminology).

	" cooccurrence " Symmetrize the matrix (undirected co-occurrence).
	"semi_markov" is <i>not</i> accepted: the package does not implement a semi-Markov / holding-time construction, so passing it errors rather than silently aliasing "tna".
directed	Logical. If TRUE (default), treat transitions as directed. If FALSE, symmetrize sequence- and edge-derived weights before returning raw/frequency weights or before row-normalizing transition probabilities.
compute_within	Logical. Compute within-cluster matrices? Default TRUE.
actor, action, time, order, session, time_threshold	Long-format event-log shortcut. When action is supplied on a data.frame input, the data is passed through prepare() to derive a wide sequence, which is then routed to the existing sequence path. Behaves identically to prepare(...) > build_network() > build_mcml().
labels	Optional name -> label remap applied to within-cluster nodes (the macro layer is left untouched because its labels are cluster names). Accepts a 2-column data.frame (name, label), a named character vector c(name = "label"), or a named list. Unmapped names pass through unchanged.

Value

A cluster_summary object with meta\$source = "transitions", fully compatible with plot(), as_tna(), and plot().

See Also

[cluster_summary](#) for matrix-based aggregation, [net_as_tna\(\)](#) to convert to tna objects, [plot\(\)](#) for visualization

Examples

```
# Edge list with clusters
edges <- data.frame(
  from = c("A", "A", "B", "C", "C", "D"),
  to   = c("B", "C", "A", "D", "D", "A"),
  weight = c(1, 2, 1, 3, 1, 2)
)
clusters <- list(G1 = c("A", "B"), G2 = c("C", "D"))
cs <- build_mcml(edges, clusters)
cs$macro$weights

# Sequence data with clusters
seqs <- data.frame(
  T1 = c("A", "C", "B"),
  T2 = c("B", "D", "A"),
  T3 = c("C", "C", "D"),
  T4 = c("D", "A", "C")
)
cs <- build_mcml(seqs, clusters, type = "raw")
cs$macro$weights
```


Description

Experimental. Aggregates a node-level psychometric network (correlation, partial correlation, or EBICglasso) into a cluster-level macro network plus per-cluster within networks - the MCML view that `build_mcml` provides for transition networks, adapted to the statistics of undirected association networks. The API and the exact aggregation formulas may change between releases.

Unlike transition counts, partial correlations do not aggregate by arithmetic: the submatrix of a pcor matrix is *not* the pcor network of the subsystem (the conditioning set changes), and averaging pcor entries across blocks is descriptive only. The five aggregation methods therefore have explicitly different statuses:

"average" (**descriptive**) Macro edge A–B = mean of the signed node-level weights between members of A and members of B; macro diagonal = mean within-block off-diagonal weight. Needs only the weight matrix (works on data-less netobjects). Caveat: signed averaging can cancel opposite-sign edges; interpret as net average association, not connectivity strength.

"composite" (**re-estimated**) Per-observation cluster scores = (optionally standardized) mean of member variables; the chosen method is then re-fit on the k composite columns, so the macro network is a genuine correlation / pcor / EBICglasso network among clusters. Requires raw data.

"loadings" (**re-estimated, connectivity-weighted**) As "composite", but member variables are weighted by their within-cluster connection strength in the node-level network (the absolute summed weight to the other members of their own cluster, normalized to sum to 1 per cluster). Nodes that anchor their cluster contribute more to its composite. This is Nestimate's own weighting - related in spirit to network loadings (Christensen & Golino 2021) but not a reimplement of any EGA-family estimator. Requires raw data.

"rv" (**descriptive, multivariate**) Macro edge A–B = Escoufier's RV coefficient between the member blocks - a matrix correlation in $[0, 1]$ computed from the block covariance structure. No composites are formed and no estimator is re-fit, so nothing is lost to averaging; signs are not represented. Requires raw data.

"canonical" (**descriptive, multivariate**) Macro edge A–B = the first canonical correlation between the member blocks: the strongest linear relationship any weighting of A's items can have with any weighting of B's items - an upper bound on what composite methods can recover. Requires raw data.

Usage

```
build_mcml_pc(
  x,
  clusters,
  aggregation = c("scaled", "composite", "mean", "median", "loadings", "average",
    "escoufier", "cancor"),
  method = c("pcor", "glasso", "cor"),
```

```

within = c("reestimate", "subnetwork"),
weighting = c("equal", "strength", "eigen", "closeness", "betweenness",
  "expected_influence", "specificity", "pca", "factor", "item_total"),
cor_method = c("pearson", "spearman", "polychoric"),
signed = TRUE,
id_col = NULL,
fa_method = c("ml", "paf", "minres", "cfa"),
...
)

```

Arguments

x	A netobject estimated with an undirected association method (cor, pcor, glasso; aliases accepted) - its \$data and \$method are reused - or a numeric data.frame of raw observations (then method decides the estimator).
clusters	Cluster membership in any of three forms: a named list of node-label vectors (names = cluster labels); a two-column data.frame read by position (first column = node names, second = group labels); or a vector of cluster labels named by node. Every node must be assigned to exactly one cluster.
aggregation	Character. How clusters are collapsed to the macro network. Default "scaled". "scaled" Re-estimate the network on per-cluster scores formed as the mean of <i>standardized</i> member items (the default; was "composite"). "composite" Backward-compatible alias for "scaled". "mean" As "scaled" but on raw (unstandardized) item means. "median" As "mean" but the per-cluster score is the row-wise median of the items (unweighted). "loadings" The scaled score path with member items weighted by their within-cluster network strength. "average" Average the item-pair edges in each between-cluster submatrix - no scores, no re-estimation. "escoufier" Escoufier RV coefficient (descriptive multivariate similarity between blocks). "cancor" First canonical correlation - an upper bound on how related two blocks can be.
method	Character. Network estimator for the re-estimation paths and for data.frame input: "pcor" (default), "glasso", or "cor" - the same vocabulary as build_network . Ignored (with the netobject's own method used instead) when x is a netobject and method is not given.
within	Character. "reestimate" (default) or "subnetwork" (see Details).
weighting	Character. How items are weighted inside their cluster score (the score paths "scaled" / "mean" / "loadings"; not "median", which is unweighted): "equal" (default) 1/m per item - the scale as scored. "strength" Mean absolute connection to the other own-cluster members in the node-level network (what aggregation = "loadings" selects).

- "eigen" Leading-eigenvector weights of the within-cluster block of the node-level network - like strength but giving extra weight to items connected to other well-connected items.
- "pca" First principal component of the member items' correlation matrix - a data-statistical weighting, blind to the estimated network.
- "factor" Standardized loadings of a one-factor model per cluster - the classical latent-variable weighting. The extraction method is chosen by `fa_method` and the correlation input respects `cor_method` (so polychoric factor analysis of ordinal items is one call). Clusters with fewer than 3 items (or non-converging fits) fall back to "pca" with a warning.
- "closeness", "betweenness" Within-cluster closeness / betweenness centrality of the item in the node-level network block (absolute weights). Betweenness can be all-zero in densely connected clusters; equal weights are used then, with a warning.
- "expected_influence" Mean *signed* connection to the other own-cluster members (Robinaugh et al. 2016) - like strength, but opposite-sign connections subtract, and negative expected influence marks reverse-keyed items.
- "specificity" The misfit margin: own-cluster strength minus the strongest cross-cluster strength, floored at 0. Items that belong as much to another cluster contribute nothing to their composite - the weighting twin of the misfit diagnostic.
- "item_total" Corrected item-total correlation: each item against the mean of the other (standardized) members - the classical scale-construction weighting.

Custom weightings: weighting also accepts a *named numeric vector* (one entry per item; absolute values are normalized within each cluster, signs flip items when `signed = TRUE`) or a *function* `function(W_block, data_block, nodes)` returning one numeric weight per item, evaluated per cluster.

Network-based and data-based weightings answer different questions; comparing them is informative - divergence means the network's view of the cluster differs from its latent-variable view. Schemes with inherently non-negative weights (equal, closeness, betweenness, specificity) keep the eigenvector-based item signs; sign-carrying schemes (eigen, pca, factor, expected_influence, item_total, custom) use their own.

<code>cor_method</code>	Character. Correlation type for estimation from raw data: "pearson" (default), "spearman", or "polychoric" (needs lavaan ; see Details).
<code>signed</code>	Logical. Flip reverse-keyed items in composites (default TRUE; see Details).
<code>id_col</code>	Character vector or NULL. Identifier column(s) to drop from <code>data.frame</code> input before analysis (e.g., the <code>rid/actor</code> columns produced by <code>convert_sequence_format(format = "frequency")</code>), whose output otherwise feeds this function directly as per-actor behavior profiles). Same convention as the association estimators.
<code>fa_method</code>	Character. Extraction method for weighting = "factor": "ml" (default) Maximum likelihood (<code>stats::factanal</code> on the <code>cor_method</code>-consistent correlation matrix). "paf" Iterated principal axis factoring (SMC start, communalities iterated on the reduced correlation matrix).

"minres" Minimum residual / unweighted least squares (uniquenesses optimized to minimize squared off-diagonal residuals).

"cfa" One-factor confirmatory model in **lavaan** (standardized loadings); with `cor_method = "polychoric"` the items are declared ordered, giving the categorical (DWLS) factor model.

On well-behaved unidimensional clusters the four agree closely; divergence indicates Heywood-prone or non-unidimensional clusters.

... Further arguments forwarded directly to the `fa_method` backend, exactly as that backend spells them (requires `weighting = "factor"`): `lavaan` arguments for "cfa" (`estimator = "WLSMV"`, `missing = "fiml"`, `se = "robust"`, ...), `stats::factanal()` arguments for "ml", and `max_iter / tol` for "paf". Arguments managed internally (`model`, `data`, `covmat`, `n.obs`, `factors`) are ignored with a warning - the model is always the one-factor model per cluster, because the composite needs exactly one weight per item.

Details

Item diagnostics. Whenever raw data or a node-level network is available, every item's connection strength to *every* cluster is computed. The `$loadings` table reports, per item: its signed own-cluster loading, its composite weight, its strongest cross-cluster loading, and a misfit flag set when the cross-cluster loading exceeds the own-cluster loading - evidence the item is assigned to the wrong cluster. Misfit items trigger a warning; every aggregation silently inherits a bad membership, so fix the assignment rather than ignoring the flag.

Reverse-keyed items. With `signed = TRUE` (default), items whose summed within-cluster association is negative are flipped (their standardized values enter composites with weight sign -1), so a reverse-keyed item reinforces its cluster composite instead of cancelling it. Flips are reported in the `sign` column and via a warning.

Missing data. Composites are per-row weighted means over the *observed* members (weights renormalized per row); rows with no observed member yield NA and are dropped by the estimator with a message. Node-level estimation applies the estimators' own complete-case handling.

Ordinal items. `cor_method = "polychoric"` (requires the **lavaan** package) estimates the node-level and within-cluster networks from polychoric correlations - appropriate for Likert items. Composites are continuous sums, so the macro re-estimation uses Pearson correlations of the composites regardless.

Within-cluster networks follow within: "reestimate" (default) re-fits the estimator on the member columns alone - the honest conditional structure of the subsystem; "subnetwork" slices the node-level weight matrix and is descriptive (for `pcor/glasso` it retains conditioning on out-of-cluster nodes). Modes without raw data force "subnetwork". Singleton clusters get no within network (NULL).

Uncertainty. The composite/loadings macro network is a full `netobject` carrying its composite data, so `bootstrap_network(fit$macro)` (edge-weight CIs) and `vertex_bootstrap(fit$macro)` (network-level CIs) work directly; `vertex_compare(fit1$macro, fit2$macro)` compares two groups. `loading_stability` quantifies how stable the composite weights themselves are under case resampling.

All constituent networks are undirected (`meta$directed = FALSE`), so renderers that auto-detect directedness (e.g. `cograph::plot_mcml()`) draw the result without arrowheads.

Value

An object of class "mcml_pc" containing:

macro Cluster-level netobject (k x k, undirected).

clusters Named list of within-cluster netobjects (NULL for singleton clusters).

cluster_members Named list of member node labels.

loadings Tidy item-diagnostic data frame (one row per node): node, cluster, loading (signed own-cluster), weight, sign, max_cross, cross_cluster, misfit. NULL only when no node-level network is available.

node_network The node-level netobject the aggregation was based on (kept for diagnostics and loading_stability).

data The raw data (data.frame) when available, else NULL.

meta List: aggregation, method, within, scale, cor_method, signed, n_nodes, n_clusters, cluster_sizes, n_misfit, n_flipped, directed = FALSE, source = "pc", experimental = TRUE.

References

- Escoufier, Y. (1973). Le traitement des variables vectorielles. *Biometrics*, 29(4), 751-760.
- Hotelling, H. (1936). Relations between two sets of variates. *Biometrika*, 28(3/4), 321-377.
- Robinaugh, D. J., Millner, A. J., & McNally, R. J. (2016). Identifying highly influential nodes in the complicated grief network. *Journal of Abnormal Psychology*, 125(6), 747-757.
- Christensen, A. P., & Golino, H. (2021). On the equivalency of factor and network loadings. *Behavior Research Methods*, 53, 1563-1580.
- Epskamp, S., & Fried, E. I. (2018). A tutorial on regularized partial correlation networks. *Psychological Methods*, 23(4), 617-634.

See Also

[build_mcml](#) for transition networks, [loading_stability](#) for composite-weight stability, [bootstrap_network](#) and [vertex_bootstrap](#) for uncertainty on the macro network.

Examples

```
set.seed(1)
n <- 200
sigma <- matrix(0.15, 6, 6)
sigma[1:3, 1:3] <- 0.5
sigma[4:6, 4:6] <- 0.5
diag(sigma) <- 1
z <- matrix(rnorm(n * 6), n, 6) %%% chol(sigma)
df <- as.data.frame(z)
names(df) <- c("a1", "a2", "a3", "b1", "b2", "b3")
clusters <- list(A = c("a1", "a2", "a3"), B = c("b1", "b2", "b3"))

fit <- build_mcml_pc(df, clusters, aggregation = "composite",
                    method = "cor")
```

```
fit$macro$weights
fit$loadings
```

```
build_mlvar
```

```
Build a Multilevel Vector Autoregression (mlVAR) network
```

Description

Estimates three networks from ESM/EMA panel data, matching `mlVAR::mlVAR()` with `estimator = "lmer"`, `temporal = "fixed"`, `contemporaneous = "fixed"` at machine precision: (1) a directed temporal network of fixed-effect lagged regression coefficients, (2) an undirected contemporaneous network of partial correlations among residuals, and (3) an undirected between-subjects network of partial correlations derived from the person-mean fixed effects.

#' @details The algorithm follows `mlVAR`'s `lmer` pipeline exactly:

1. Drop rows with NA in `id/day/beep` and optionally grand-mean standardize each variable.
2. Expand the `per-(id, day)` beep grid and right-join original values, producing the augmented panel (`augData`).
3. Add within-person lagged predictors (`L1_*`) and person-mean predictors (`PM_*`).
4. For each outcome variable fit `lmer(y ~ within + between-except-own-PM + (1 | id))` with `REML = FALSE`. Collect the fixed-effect temporal matrix `B`, between-effect matrix `Gamma`, random-intercept SDs (`mu_SD`), and `lmer` residual SDs.
5. Contemporaneous network: `cor2pcor(D %%% cov2cor(cor(resid)) %%% D)`.
6. Between-subjects network: `cor2pcor(pseudoinverse(forcePositive(D (I - Gamma))))`.

Validated to machine precision (`max_diff < 1e-10`) against `mlVAR::mlVAR()` on 25 real ESM datasets from `openesm` and 20 simulated configurations (seeds 201-220). See `tmp/mlvar_equivalence_real20.R` and `tmp/mlvar_equivalence_20seeds.R`.

Usage

```
build_mlvar(
  data,
  vars,
  id,
  day = NULL,
  beep = NULL,
  lag = 1L,
  standardize = FALSE
)
```

Arguments

data	A data.frame containing the panel data.
vars	Character vector of variable column names to model.
id	Character string naming the person-ID column.
day	Character string naming the day/session column, or NULL. When provided, lag pairs are only formed within the same day.
beep	Character string naming the measurement-occasion column, or NULL. When NULL, row position within each (id, day) is used.
lag	Integer. The lag order (default 1).
standardize	Logical. If TRUE, each variable is grand-mean centered and divided by its pooled SD <i>before</i> augmentation. Default FALSE, matching mlVAR::mlVAR(scale = FALSE) - the only setting for which numerical equivalence has been validated.

Value

A dual-class `c("net_mlvar", "netobject_group")` object - a named list of three full netobjects, one per network, plus model-level metadata stored as attributes. Each element is a standard `c("netobject", "cograph_network")` weight-matrix wrapper (no raw \$data), so `print()`, `summary()`, `coefs()`, and `cograph::splot(fit$temporal)` work directly. The three constituents are matrix-wrapped and carry no underlying panel data, so data-resampling verbs such as `bootstrap_network()` (and reliability/stability) cannot iterate over them - extract a single constituent and rebuild via `build_network()` if you need those. Structure:

`fit$temporal` Directed netobject for the $d \times d$ matrix of fixed-effect lagged coefficients. `$weights[i, j]` is the effect of variable `j` at `t`-lag on variable `i` at `t`. `method = "mlvar_temporal", directed = TRUE`.

`fit$contemporaneous` Undirected netobject for the $d \times d$ partial-correlation network of within-person lmer residuals. `method = "mlvar_contemporaneous", directed = FALSE`.

`fit$between` Undirected netobject for the $d \times d$ partial-correlation network of person means, derived from $D(I - \Gamma)$. `method = "mlvar_between", directed = FALSE`.

`attr(fit, "coefs") / coefs()` Tidy data.frame with one row per (outcome, predictor) pair and columns outcome, predictor, beta, se, t, p, ci_lower, ci_upper, significant. Filter, sort, or plot with base R or the tidyverse. Retrieve with `coefs(fit)`.

`attr(fit, "n_obs")` Number of rows in the augmented panel after `na.omit`.

`attr(fit, "n_subjects")` Number of unique subjects remaining.

`attr(fit, "lag")` Lag order used.

`attr(fit, "standardize")` Logical; whether pre-augmentation standardization was applied.

See Also

[build_network\(\)](#)

Examples

```
## Not run:
d <- simulate_data("mlvar", seed = 1)
fit <- build_mlvar(d, vars = attr(d, "vars"),
                  id = "id", day = "day", beep = "beep")

print(fit)
summary(fit)

## End(Not run)
```

 build_mmm

Fit a Mixed Markov Model

Description

Discovers latent subgroups with different transition dynamics using Expectation-Maximization. Each mixture component has its own transition matrix. Sequences are probabilistically assigned to components.

Usage

```
build_mmm(
  data,
  k = 2L,
  n_starts = 50L,
  max_iter = 200L,
  tol = 1e-06,
  smooth = 0.01,
  seed = NULL,
  covariates = NULL,
  covariate_effect = c("em", "posthoc"),
  estimator = c("auto", "firth", "multinom", "chisq")
)
```

Arguments

data	A data.frame (wide format), netobject, or tna model. For tna objects, extracts the stored data.
k	Integer. Whole finite number of mixture components, ≥ 2 . Default: 2.
n_starts	Integer. Positive whole finite number of random restarts. Default: 50.
max_iter	Integer. Positive whole finite maximum EM iterations per start. Default: 200.
tol	Numeric. Finite positive convergence tolerance. Default: 1e-6.
smooth	Numeric. Finite non-negative Laplace smoothing constant. Default: 0.01.
seed	Integer or NULL. Random seed.

covariates	Optional. Covariates integrated into the EM algorithm to model covariate-dependent mixing proportions. Accepts a string, character vector, formula, or data.frame (same forms as build_clusters). For netobject or cograph_network input, names are resolved against \$metadata first, so a typical call is build_mmm(net, k = 3, covariates = "session_label"). Unlike the post-hoc analysis in build_clusters(), these covariates directly influence cluster membership during EM estimation (see covariate_effect).
covariate_effect	How covariates enter the model. "em" (default) folds them into the EM as covariate-dependent mixing proportions, so they shape the cluster fit itself (and rows with missing covariates are dropped before fitting). "posthoc" fits a plain mixture on every sequence and uses the covariates only for the after-fit multinomial logit, so covariate values — and their missingness — never change which clusters are found. Ignored when covariates is NULL.
estimator	Multinomial fitter for the post-hoc covariate analysis (does not affect EM): "auto" (default) inspects the cluster x covariate cross-tab and falls back to "firth" only when any cell has fewer than 5 observations (separation risk), otherwise the much faster "multinom"; "firth" forces Firth's penalised likelihood via brglm2::brmultinom (finite under separation); "multinom" forces nnet::multinom (warns about separation risk); "chisq" runs descriptive tests (no logit). See build_clusters for full details.

Value

An object of class net_mmm with components:

data The full N-row sequence frame used for estimation.

models List of netobjects, one per component. Each component carries the rows assigned to that component in its \$data slot, while its transition matrix is the EM-estimated component transition matrix.

k Number of components.

mixing Numeric vector of mixing proportions.

posterior N x k matrix of posterior probabilities.

assignments Integer vector of hard assignments (1..k).

quality List: avepp (per-class), avepp_overall, entropy, relative_entropy, classification_error.

log_likelihood, BIC, AIC, ICL Model fit statistics.

states Character vector of state names.

Initial states

The first sequence column has special status: it is read directly as the per-sequence initial state (`init_state[i] <- match(raw_data[i, state_cols[1L]], states)`). The function does **not** scan forward to the first non-missing position, and it does not apply any na_syms-style symbol conversion (unlike [build_clusters](#)). The state vocabulary is built from the unique non-NA values across all columns, so if your data uses a sentinel character such as "*" or "%" for missing cells, that sentinel becomes a real state and the first column reads it as a valid initial state. If you want padded

leading missings to be treated as missing, recode them to NA before calling `build_mmm()` (then `match()` returns NA, which the EM treats as an uninformative initial distribution), or left-trim the leading missings so each sequence's first column carries an observed state.

See Also

[compare_mmm](#), [build_network](#)

Examples

```
seqs <- data.frame(V1 = sample(c("A", "B", "C"), 30, TRUE),
                   V2 = sample(c("A", "B", "C"), 30, TRUE))
mmm <- build_mmm(seqs, k = 2, n_starts = 1, max_iter = 10, seed = 1)
mmm

seqs <- data.frame(
  V1 = sample(LETTERS[1:3], 30, TRUE), V2 = sample(LETTERS[1:3], 30, TRUE),
  V3 = sample(LETTERS[1:3], 30, TRUE), V4 = sample(LETTERS[1:3], 30, TRUE)
)
mmm <- build_mmm(seqs, k = 2, seed = 42)
print(mmm)
summary(mmm)
```

build_mogen

Build Multi-Order Generative Model (MOGen)

Description

Constructs higher-order De Bruijn graphs from sequential trajectory data and selects the optimal Markov order using AIC, BIC, or likelihood ratio tests.

Usage

```
build_mogen(
  data,
  max_order = 5L,
  criterion = c("aic", "bic", "lrt"),
  lrt_alpha = 0.01
)
```

Arguments

data	A data.frame (rows = trajectories, columns = time points) or a list of character/numeric vectors (one per trajectory).
max_order	Integer. Maximum Markov order to test (default 5). Must be a whole number; a non-integer value (e.g. 2.7) is an error rather than being silently truncated.

criterion	Character. Model selection criterion: "aic" (default), "bic", or "lrt" (likelihood ratio test).
lrt_alpha	Numeric. Significance threshold for LRT (default 0.01).

Details

At order k , nodes are k -tuples of states and edges represent transitions between overlapping k -tuples. The model tests increasingly complex Markov orders and selects the one that best balances fit and parsimony.

Value

An object of class `net_mogen` with components:

optimal_order Selected optimal Markov order.
criterion Which criterion was used for selection.
orders Integer vector of tested orders (0 to `max_order`).
aic Named numeric vector of AIC values per order.
bic Named numeric vector of BIC values per order.
log_likelihood Named numeric vector of log-likelihoods.
dof Named integer vector of cumulative DOF per model.
layer_dof Named integer vector of per-layer DOF.
transition_matrices List of transition matrices (index 1 = order 0).
states Unique first-order states.
n_paths Number of trajectories.
n_observations Total number of state observations.

References

Scholtes, I. (2017). When is a Network a Network? Multi-Order Graphical Model Selection in Pathways and Temporal Networks. *KDD 2017*.
 Gote, C. & Scholtes, I. (2023). Predicting variable-length paths in networked systems using multi-order generative models. *Applied Network Science*, 8, 62.

Examples

```
seqs <- list(c("A","B","C","D"), c("A","B","C","A"), c("B","C","D","A"))
mg <- build_mogen(seqs, max_order = 2)

trajs <- list(c("A","B","C","D"), c("A","B","D","C"),
             c("B","C","D","A"), c("C","D","A","B"))
m <- build_mogen(trajs, max_order = 3)
print(m)
plot(m)
```

build_network

*Build a Network***Description**

Universal network estimation function that supports both transition networks (relative, frequency, co-occurrence) and association networks (correlation, partial correlation, graphical lasso). Uses the global estimator registry, so custom estimators can also be used.

Usage

```
build_network(
  data,
  method,
  actor = NULL,
  action = NULL,
  time = NULL,
  session = NULL,
  order = NULL,
  codes = NULL,
  group = NULL,
  format = "auto",
  window_size = 3L,
  mode = c("non-overlapping", "overlapping"),
  scaling = NULL,
  threshold = 0,
  level = NULL,
  time_threshold = 900,
  predictability = TRUE,
  state_cols = NULL,
  metadata_cols = NULL,
  start = FALSE,
  end = FALSE,
  params = list(),
  labels = NULL,
  ...
)
```

Arguments

data	Data frame (sequences or per-observation frequencies) or a square symmetric matrix (correlation or covariance).
method	Character. Required. Name of a registered estimator. Built-in methods: "relative", "frequency", "co_occurrence", "cor", "pcor", "glasso", "ising", "mgm", "attention", "wtna", "wtna_cooccurrence". Aliases: "tna" and "transition" map to "relative"; "ftna" and "counts" map to "frequency"; "cna" and "wcna" map to "co_occurrence"; "corr" and "correlation" map to "cor";

"partial" maps to "pcor"; "ebicglasso" and "regularized" map to "glasso"; "isingfit" maps to "ising"; "atna" maps to "attention"; "mixed" and "mixed_graphical" map to "mgm"; "wtna_transition" maps to "wtna".

actor	Character. Name of the actor/person ID column for sequence grouping. Default: NULL.
action	Character. Name of the action/state column (long format). Default: NULL.
time	Character. Name of the time column (long format). Default: NULL.
session	Character. Name of the session column. Default: NULL.
order	Character. Name of the ordering column. Default: NULL.
codes	Character vector. Column names of one-hot encoded states (for onehot format). Default: NULL.
group	Character. Name of a grouping column for per-group networks. Returns a netobject_group (named list of netobjects). Default: NULL.
format	Character. Input format: "auto", "wide", "long", or "onehot". Default: "auto".
window_size	Integer. Window size for one-hot windowing. Default: 3L.
mode	Character. Windowing mode for one-hot input only: "non-overlapping" or "overlapping". Has no effect on wide or long sequence data (only the one-hot/wtna path reads it). Default: "non-overlapping".
scaling	Character vector or NULL. Post-estimation scaling to apply (in order). Options: "minmax", "max", "rank", "normalize". Can combine: c("rank", "minmax"). Default: NULL (no scaling).
threshold	Numeric. Absolute values below this are set to zero in the result matrix. Default: 0 (no thresholding).
level	Character or NULL. Multilevel decomposition for association methods. One of NULL, "between", "within", "both". Requires id_col. Default: NULL.
time_threshold	Numeric. Maximum time gap (seconds) for long format session splitting. Default: 900.
predictability	Logical. If TRUE (default), compute and store node predictability (R-squared) for undirected association methods (glasso, pcor, cor). Stored in \$predictability and auto-displayed as donuts by cograph::splot().
state_cols	Character vector or NULL. Explicit names of columns to classify as state columns in the returned netobject's \$data slot. When provided, all other columns of the cleaned input go to \$metadata. Auto-detection (values-in-nodes heuristic) is bypassed. Use this when a metadata column happens to contain values that overlap with node names (e.g. condition labels "A", "B", "C" and nodes "A", "B", "C") and auto-detection would misclassify it. Default: NULL (auto-detect).
metadata_cols	Character vector or NULL. Explicit names of columns to force into the \$metadata slot. The remaining columns are auto-detected as state via the values-in-nodes rule. Cannot overlap with state_cols. Default: NULL.

start	Boundary marker prepended to every sequence as an explicit start state (a pure source: no incoming edges, every sequence's first transition is start -> first_observed). FALSE (default) adds nothing; TRUE uses the label "Start"; a single string uses that string as the label. Only valid for the transition methods (relative, frequency, co_occurrence, attention); errors otherwise.
end	Boundary marker placed in the single cell after each sequence's last observed (non-NA) state, as an explicit terminal state (a pure sink: no outgoing edges, no self-loop – distinct from <code>mark_terminal_state</code> , which fills all trailing NAs into an absorbing state). FALSE (default) adds nothing; TRUE uses the label "End"; a single string uses that string as the label. Same method restriction as start.
params	Named list. Method-specific parameters passed to the estimator function (e.g. <code>list(gamma = 0.5)</code> for <code>glasso</code> , or <code>list(format = "wide")</code> for transition methods). This is the key composability feature: downstream functions like bootstrap or grid search can store and replay the full params list without knowing method internals. Transition estimators accept tna-style sequence options such as <code>weighted</code> and <code>concat</code> (and the low-level <code>begin_state</code> / <code>end_state</code> , of which <code>start</code> / <code>end</code> are the public form – see those arguments). Column-like entries in params (<code>action</code> , <code>id</code> , <code>id_col</code> , <code>time</code> , <code>session</code> , <code>order</code> , <code>codes</code> , and <code>group</code>) are resolved before format detection and must name existing columns. If the same column role is supplied both directly and through params, the names must agree.
labels	Optional name -> label remap applied after construction. Accepts a 2-column <code>data.frame</code> (<code>name</code> , <code>label</code>), a named character vector <code>c(name = "label")</code> , or a named list. Rewrites <code>\$nodes\$label</code> and <code>dimnames(weights)</code> . Unmapped names pass through unchanged.
...	Additional arguments passed to the estimator function.

Details

The function works as follows:

1. Resolves method aliases to canonical names.
2. Validates explicit column arguments before any format guessing.
3. Retrieves the estimator function from the global registry.
4. For association methods with `level` specified, decomposes the data (between-person means or within-person centering).
5. Calls the estimator: `do.call(fn, c(list(data = data), params))`.
6. Applies scaling and thresholding to the result matrix.
7. Extracts edges and constructs the `netobject`.

For long-format transition data, supplying `action` without `actor` is allowed and treats all rows as one sequence in row/time order. The function warns because a one-sequence transition network is not recommended and cannot be validated by bootstrap or other confirmatory tests.

Value

An object of class `c("netobject", "cograph_network")` containing:

data The input data used for estimation, as a data frame.

weights The estimated network weight matrix.

nodes Data frame with columns `id`, `label`, `name`, `x`, `y`. Node labels are in `$nodes$label`.

edges Data frame of non-zero edges with integer `from/to` (node IDs) and numeric `weight`.

directed Logical. Whether the network is directed.

method The resolved method name.

params The params list used (for reproducibility).

scaling The scaling applied (or `NULL`).

threshold The threshold applied.

n_nodes Number of nodes.

n_edges Number of non-zero edges.

level Decomposition level used (or `NULL`).

meta List with source, layout, and tna metadata (cograph-compatible).

node_groups Node groupings data frame, or `NULL`.

predictability Named numeric vector of R-squared predictability values per node (for undirected association methods when `predictability = TRUE`). `NULL` for directed methods.

Method-specific extras (e.g. `precision_matrix`, `cor_matrix`, `frequency_matrix`, `lambda_selected`, etc.) are preserved from the estimator output.

When `level = "both"`, returns an object of class `"netobject_m1"` with `$between` and `$within` sub-networks and a `$method` field.

See Also

[register_estimator](#), [list_estimators](#), [bootstrap_network](#)

Examples

```
seqs <- data.frame(V1 = c("A", "B", "C", "A"), V2 = c("B", "C", "A", "B"))
net <- build_network(seqs, method = "relative")
net

# Transition network (relative probabilities)
seqs <- data.frame(
  V1 = sample(LETTERS[1:4], 30, TRUE), V2 = sample(LETTERS[1:4], 30, TRUE),
  V3 = sample(LETTERS[1:4], 30, TRUE), V4 = sample(LETTERS[1:4], 30, TRUE)
)
net <- build_network(seqs, method = "relative")
print(net)

# Association network (glasso)
freq_data <- convert_sequence_format(seqs, format = "frequency")
net_glasso <- build_network(freq_data, method = "glasso",
```

```

      params = list(gamma = 0.5, nlambda = 50))

# With scaling
net_scaled <- build_network(seqs, method = "relative",
                           scaling = c("rank", "minmax"))

```

 build_pcor

Build a Partial Correlation Network

Description

Convenience wrapper for `build_network(method = "pcor")`. Computes partial correlations from numeric data.

Usage

```
build_pcor(data, ...)
```

Arguments

<code>data</code>	Data frame (sequences or per-observation frequencies) or a square symmetric matrix (correlation or covariance).
<code>...</code>	Additional arguments passed to build_network .

Value

A `netobject` (see [build_network](#)).

See Also

[build_network](#)

Examples

```

data(srl_strategies)
net <- build_pcor(srl_strategies)

```

build_simplicial	<i>Build a Simplicial Complex</i>
------------------	-----------------------------------

Description

Constructs a simplicial complex from a network or higher-order pathway object. Two construction methods are available:

- **Clique complex** ("clique"): every clique in the thresholded non-zero graph becomes a simplex. Edges with absolute weight $\geq$ threshold are retained. The standard bridge from graph theory to algebraic topology.
- **Pathway complex** ("pathway"): each higher-order pathway from a net_hon or net_hypa becomes a simplex.

For type = "vr" (or alias "rips"), the input is treated as a non-negative distance / dissimilarity matrix and a Vietoris-Rips filtration is constructed: each k-simplex σ enters at $\max_{(i,j) \in \sigma} d(i,j)$. Use max_scale to cap the filtration diameter; edges with $d(i,j) > \text{max_scale}$ are excluded. Filtration values are attached as \$filtration on the returned object so persistent_homology() can read them directly.

Usage

```
build_simplicial(
  x,
  type = "clique",
  threshold = 0,
  max_dim = 10L,
  max_pathways = NULL,
  anomaly = c("all", "over", "under"),
  max_scale = NULL,
  ...
)
```

Arguments

x	A square matrix, tna, netobject, net_hon, net_hypa, or net_mogen.
type	Construction type: "clique" (default), "pathway", or "vr" (alias "rips").
threshold	For type = "clique": minimum non-zero absolute edge weight to include an edge (default 0). Edges below this are ignored; zero-weight non-edges are never included. Ignored for type = "vr" - use max_scale instead.
max_dim	Maximum simplex dimension (default 10). Must be a single non-negative integer. A k-simplex has k+1 nodes.
max_pathways	For type = "pathway": maximum number of pathways to include, ranked by count (HON) or ratio (HYPA). NULL includes all. Default NULL.

anomaly	For HYPA pathway complexes, which anomaly direction to include: "all" (default), "over", or "under". Under-represented HYPA paths are ranked by smallest observed/expected ratio; over-represented paths are ranked by largest ratio.
max_scale	For type = "vr": maximum edge length to include in the filtration. NULL (default) uses max(d).
...	Additional arguments passed to build_hon() when x is a tna/netobject with type = "pathway".

Value

A simplicial_complex object. For type = "vr" an additional \$filtration numeric vector is attached (parallel to \$simplices).

See Also

[betti_numbers](#), [persistent_homology](#), [simplicial_degree](#), [q_analysis](#)

Examples

```
mat <- matrix(c(0,.6,.5,.6,0,.4,.5,.4,0), 3, 3)
colnames(mat) <- rownames(mat) <- c("A","B","C")
sc <- build_simplicial(mat, threshold = 0.3)
print(sc)
betti_numbers(sc)

# Vietoris-Rips on a distance matrix:
d <- 1 - mat
diag(d) <- 0
sc_vr <- build_simplicial(d, type = "vr", max_scale = 0.6)
```

build_tna	<i>Build a Transition Network (TNA)</i>
-----------	---

Description

Convenience wrapper for build_network(method = "relative"). Computes row-normalized transition probabilities from sequence data.

Usage

```
build_tna(data, start = FALSE, end = FALSE, ...)
```

Arguments

data	Data frame (sequences or per-observation frequencies) or a square symmetric matrix (correlation or covariance).
start	Boundary marker prepended to every sequence as an explicit start state (a pure source: no incoming edges, every sequence's first transition is start -> first_observed). FALSE (default) adds nothing; TRUE uses the label "Start"; a single string uses that string as the label. Only valid for the transition methods (relative, frequency, co_occurrence, attention); errors otherwise.
end	Boundary marker placed in the single cell after each sequence's last observed (non-NA) state, as an explicit terminal state (a pure sink: no outgoing edges, no self-loop – distinct from mark_terminal_state , which fills all trailing NAs into an absorbing state). FALSE (default) adds nothing; TRUE uses the label "End"; a single string uses that string as the label. Same method restriction as start.
...	Additional arguments passed to build_network .

Value

A netobject (see [build_network](#)).

See Also

[build_network](#)

Examples

```
seqs <- data.frame(V1 = c("A", "B", "C"), V2 = c("B", "C", "A"))
net <- build_tna(seqs)
```

`casedrop_reliability` *Edge-weight Case-dropping Stability*

Description

Computes a **CS-coefficient for the edge-weight vector** of a network: the maximum proportion of cases (rows of `x$data`) that can be dropped while the flattened edge-weight vector of the re-estimated network still correlates with the original above threshold in at least certainty of iterations.

Plots the four model-level reliability metrics across drop proportions: correlation, mean_abs_dev, median_abs_dev, max_abs_dev. Each panel shows the per-iteration mean with a ribbon at mean +/- sd. The correlation panel includes a dashed horizontal line at the user's threshold (default 0.7).

Overlay of per-cluster correlation curves across drop proportions. One colour per sub-network; ribbons show mean +/- sd across iterations. Dashed horizontal line marks the stability threshold (default 0.7).

Usage

```

casedrop_reliability(
  x,
  iter = 1000L,
  drop_prop = seq(0.1, 0.9, by = 0.1),
  threshold = 0.7,
  certainty = 0.95,
  method = c("spearman", "pearson", "kendall"),
  include_diag = FALSE,
  seed = NULL
)

## S3 method for class 'net_casedrop_reliability'
print(x, digits = 3, ...)

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

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

## S3 method for class 'net_casedrop_reliability_group'
summary(object, drop_prop = NULL, ...)

## S3 method for class 'summary.net_casedrop_reliability_group'
print(x, ...)

## S3 method for class 'net_casedrop_reliability'
plot(x, combined = TRUE, ...)

## S3 method for class 'net_casedrop_reliability_group'
plot(
  x,
  metric = c("correlation", "mean_abs_dev", "median_abs_dev", "max_abs_dev"),
  ...
)

```

Arguments

<code>x</code>	A <code>net_casedrop_reliability_group</code> object.
<code>iter</code>	Integer. Iterations per drop proportion. Default 1000.
<code>drop_prop</code>	Drop proportion at which to report the four metrics (mean +/- sd per network). Must be one of the drop proportions the object was built with. Defaults to the object's median grid value (the stored grid is used, not an assumed 0.7); pass an explicit value not in the grid to get an error listing the available proportions.
<code>threshold</code>	Numeric in $[0, 1]$. Minimum edge-vector correlation for an iteration to count as stable. Default 0.7.

certainty	Numeric in $[0, 1]$. Required fraction of iterations whose correlation must exceed threshold for a drop proportion to qualify. Default 0.95.
method	Correlation method: "pearson" (weight magnitudes), "spearman" (ranks, robust to scale), or "kendall". Default "spearman" because edge weights often span several orders of magnitude and rank stability is the typical target.
include_diag	Logical. Include diagonal (self-loop) edges in the edge vector. Default FALSE.
seed	Optional integer for reproducibility.
digits	Digits to display. Default 3.
...	Additional arguments (ignored).
object	A <code>net_casedrop_reliability_group</code> .
combined	When TRUE (default), all four metrics are shown in one ggplot via <code>facet_wrap(~metric)</code> . When FALSE, returns a named list of four single-panel ggplots, one per metric.
metric	Which metric to plot. One of "correlation" (default), "mean_abs_dev", "median_abs_dev", "max_abs_dev".

Details

Complements `centrality_stability()`: that function asks whether centrality *rankings* are stable; this one asks whether the *edge-weight structure itself* is stable. For MCML-derived networks where each row of `$data` is one transition, this is case-dropping of **edges**.

For each drop_prop `p` and each iteration, a size `n_cases * (1 - p)` subset of `$data` rows is selected **without replacement**, the network is re-estimated using the same method/scaling/threshold as the input, and the upper/lower-triangle (directed: all off-diagonal entries) of the new weight matrix is flattened and correlated with the corresponding vector of the original matrix. The correlation method defaults to Spearman for robustness to the wide dynamic range of transition probabilities.

Unlike bootstrap CIs, case-dropping does not estimate sampling variance and so does not rely on the i.i.d. assumption. This makes it the appropriate robustness check for **edgelist-derived** networks (where rows of `$data` lack actor grouping), since dropping rows at random is a well-posed operation regardless of within-actor correlation.

Value

An object of class `net_casedrop_reliability` with:

`cs` Scalar CS-coefficient - the maximum drop proportion for which the edge-vector correlation remains $\geq$ threshold in at least `certainty` of iterations. Zero if no proportion qualifies.

`correlations` `iter x length(drop_prop)` matrix of per- iteration correlations.

`drop_prop`, `threshold`, `certainty`, `iter`, `method` Inputs.

The input `x` invisibly.

A tidy data frame with columns `metric`, `drop_prop`, `mean`, `sd` summarising edge-weight stability across case-dropping iterations.

A data frame with one row per network containing `cor`, `mean_abs_dev`, `median_abs_dev`, `max_abs_dev` formatted as "mean +/- sd".

A ggplot object, or a named list of four ggplots when `combined = FALSE`.

A ggplot object.

References

Epskamp, S., Borsboom, D., & Fried, E. I. (2018). Estimating psychological networks and their accuracy: A tutorial paper. *Behavior Research Methods* 50(1), 195-212. doi:10.3758/s13428017-08621

See Also

`centrality_stability()`, `bootstrap_network()`.

Examples

```
seqs <- data.frame(
  V1 = sample(LETTERS[1:4], 30, TRUE),
  V2 = sample(LETTERS[1:4], 30, TRUE),
  V3 = sample(LETTERS[1:4], 30, TRUE)
)
net <- build_network(seqs, method = "relative")
es <- casedrop_reliability(net, iter = 50, drop_prop = c(0.1, 0.3, 0.5),
  seed = 1)
print(es)
```

centrality_stability *Centrality Stability Coefficient (CS-coefficient)*

Description

Estimates the stability of centrality indices under case-dropping. For each drop proportion, sequences are randomly removed and the network is re-estimated. The correlation between the original and subset centrality values is computed. The CS-coefficient is the maximum proportion of cases that can be dropped while maintaining a correlation above threshold in at least certainty of bootstrap samples.

For transition methods, uses pre-computed per-sequence count matrices for fast resampling. Strength centralities (InStrength, OutStrength) are computed directly from the matrix without igrph.

Usage

```
centrality_stability(
  x,
  measures = c("InStrength", "OutStrength", "Betweenness"),
  iter = 1000L,
  drop_prop = seq(0.1, 0.9, by = 0.1),
  threshold = 0.7,
  certainty = 0.95,
  method = "pearson",
  centrality_fn = NULL,
  loops = FALSE,
```

```

    normalize = FALSE,
    invert = TRUE,
    normalize_diffusion = TRUE,
    seed = NULL
  )

```

Arguments

<code>x</code>	A netobject from build_network .
<code>measures</code>	Character vector. Centrality measures to assess. Defaults to <code>c("InStrength", "OutStrength", "Betweenness")</code> . Pass "all" for every built-in measure: "OutStrength", "InStrength", "ClosenessIn", "ClosenessOut", "Closeness", "Betweenness", "BetweennessRSP", "Diffusion", and "Clustering". The legacy aliases "InCloseness" and "OutCloseness" are also accepted. Custom measures beyond these are valid only when a <code>centrality_fn</code> is supplied to resolve them.
<code>iter</code>	Integer. Number of bootstrap iterations per drop proportion (default: 1000).
<code>drop_prop</code>	Numeric vector. Proportions of cases to drop (default: <code>seq(0.1, 0.9, by = 0.1)</code>).
<code>threshold</code>	Numeric. Minimum correlation to consider stable (default: 0.7).
<code>certainty</code>	Numeric. Required proportion of iterations above threshold (default: 0.95).
<code>method</code>	Character. Correlation method: "pearson", "spearman", or "kendall" (default: "pearson").
<code>centrality_fn</code>	Optional function. A custom centrality function that takes a weight matrix and returns a named list of centrality vectors. When NULL (default), all built-in measures are computed internally: "InStrength"/"OutStrength" via <code>colSums/rowSums</code> , and "Betweenness"/"InCloseness"/"OutCloseness"/"Closeness" via an internal Floyd-Warshall shortest-path routine. When provided, the function is called as <code>centrality_fn(mat)</code> and is used only for requested measures that are not one of the six built-ins; it should return a named list (e.g., <code>list(my_metric = ...)</code>).
<code>loops</code>	Logical. If FALSE (default), self-loops (diagonal) are excluded from centrality computation. This does not modify the stored matrix.
<code>normalize</code>	Logical. Range-normalize all requested measures using the same transformation as <code>tna::centralities(normalize = TRUE)</code> . Default: FALSE.
<code>invert</code>	Logical. Invert weights for shortest-path measures? Default: TRUE, matching <code>tna</code> .
<code>normalize_diffusion</code>	Logical. Range-normalize Diffusion even when <code>normalize = FALSE</code> . Default: TRUE.
<code>seed</code>	Integer or NULL. RNG seed for reproducibility.

Value

An object of class "net_stability" containing:

cs Named numeric vector of CS-coefficients per measure.

correlations Named list of matrices (iter x n_prop) of correlation values per measure.

measures Character vector of measures assessed.

drop_prop Drop proportions used.

threshold Stability threshold.

certainty Required certainty level.

iter Number of iterations.

method Correlation method.

See Also

[build_network](#), [network_reliability](#)

Examples

```
net <- build_network(data.frame(V1 = c("A","B","C","A"),
  V2 = c("B","C","A","B")), method = "relative")
cs <- centrality_stability(net, iter = 10, drop_prop = 0.3)

seqs <- data.frame(
  V1 = sample(LETTERS[1:4], 30, TRUE), V2 = sample(LETTERS[1:4], 30, TRUE),
  V3 = sample(LETTERS[1:4], 30, TRUE), V4 = sample(LETTERS[1:4], 30, TRUE)
)
net <- build_network(seqs, method = "relative")
cs <- centrality_stability(net, iter = 100, seed = 42,
  measures = c("InStrength", "OutStrength"))
print(cs)
```

certainty

Analytic certainty of network edges (Bayesian Dirichlet-Multinomial)

Description

Closed-form alternative to [bootstrap_network](#) for transition networks. Models the outgoing transitions from each state as a Dirichlet-Multinomial process: with a Jeffreys prior the posterior for state i is $\text{Dirichlet}(c_i + \text{prior})$, so each edge is marginally Beta and its posterior mean, standard deviation, credible interval and stability decision are available analytically. No resampling, so it runs in microseconds.

The return value has the same structure as [bootstrap_network](#) (same slots and summary columns) and carries class `c("net_certainty", "net_bootstrap")`, so `summary()` and any code that consumes a `net_bootstrap` object work unchanged.

Usage

```
certainty(
  x,
  prior = 0.5,
  ci_level = 0.05,
  inference = c("stability", "threshold"),
  consistency_range = c(0.75, 1.25),
  edge_threshold = NULL
)
```

Arguments

<code>x</code>	A netobject from build_network using a transition-probability method ("relative" / "tna"), or a netobject_group.
<code>prior</code>	Numeric. Dirichlet prior concentration added to every cell (default 0.5, the Jeffreys prior).
<code>ci_level</code>	Numeric in (0,1). Tail level for credible intervals and the stability decision (default 0.05, i.e. a 95\ match <code>bootstrap_network()</code>).
<code>inference</code>	Character. "stability" (default) tests whether the posterior keeps the edge within a multiplicative <code>consistency_range</code> of its weight; "threshold" tests whether the edge exceeds <code>edge_threshold</code> .
<code>consistency_range</code>	Numeric vector of length 2. Multiplicative bounds for stability inference (default <code>c(0.75, 1.25)</code>).
<code>edge_threshold</code>	Numeric or NULL. Fixed threshold for inference = "threshold". If NULL, defaults to the 10th percentile of non-zero edge weights.

Details

Certainty (this function), stability ([bootstrap_network](#)) and reliability ([reliability](#)) answer different questions about an edge: how precisely it is pinned down by the observed counts, whether it survives resampling the sequences, and whether it is consistent across split-halves. Certainty and stability agree on homogeneous data; certainty is over-confident when the data are a mixture of latent classes, because it treats transitions clustered within a sequence as independent.

Value

An object of class `c("net_certainty", "net_bootstrap")` with the same fields as [bootstrap_network](#): `original`, `mean`, `sd`, `p_values`, `significant`, `ci_lower`, `ci_upper`, `cr_lower`, `cr_upper`, `summary`, `model`, `method`, `params`, `ci_level`, `inference`, `consistency_range`, `edge_threshold`, plus `prior` and `iter = NA` (no iterations).

References

Johnston, L. & Jendoubi, T. (2026). How Delivery Mode Reshapes Resource Engagement: A Bayesian Differential Network Analysis. TNA Workshop 2026.

See Also

[bootstrap_network](#), [bayes_compare](#), [network_reliability](#)

Examples

```
seqs <- data.frame(V1 = c("A", "B", "A", "C", "B"), V2 = c("B", "C", "B", "A", "C"),
                   V3 = c("C", "A", "C", "B", "A"))
net <- build_network(seqs, method = "relative")
cert <- certainty(net)
cert
summary(cert)
```

chain_structure

Qualitative structure of a discrete-time Markov chain

Description

Computes properties that depend only on the transition matrix support, not on any starting distribution: state classification, communicating classes, periods, irreducibility / aperiodicity / regularity / reversibility, hitting probabilities, and absorption analysis when absorbing states exist.

Usage

```
chain_structure(x, normalize = TRUE, tol = 1e-10)
```

Arguments

x	A netobject, cograph_network, tna model, transition matrix, or sequence data.frame (passed through build_network() with method = "relative").
normalize	Logical. If TRUE (default), rows of the transition matrix are renormalized to sum to 1 before analysis (see passage_time() for the same convention).
tol	Numerical tolerance for the reversibility check (detailed balance) and for treating near-zero entries as zero when building the support graph (which drives classification, communicating_classes, period, and hitting_probabilities). It does not govern the absorbing-state test: a state is absorbing only when $P[i, i]$ equals 1 to an internal fixed tolerance of $.Machine$double.eps^{0.5}$, independent of tol (so raising tol to ignore tiny transition probabilities never reclassifies a near-deterministic state as absorbing). Default 1e-10.

Details

Built specifically as a diagnostic to run *before* trusting the output of [passage_time\(\)](#) or [markov_stability\(\)](#). Both implicitly assume a regular chain (irreducible + aperiodic) so that the stationary distribution is unique and meaningful. Use is_regular to check.

The fundamental-matrix absorption math follows Kemeny & Snell (1976); the hitting-probability linear system follows Norris (1997).

Value

A chain_structure object: a list with elements

states Character vector of state names.

classification Named character vector. One of "absorbing", "recurrent", "transient" per state.

communicating_classes List of state-name vectors. Each sublist is a strongly connected component of the support graph.

recurrent_classes Subset of communicating_classes that are closed (no transitions leaving the class).

transient_classes Subset that are not closed.

absorbing_states Character vector of states with $P[i, i] = 1$ (tested exactly, to within `.Machine$double.eps^0.5`; the user-facing `tol` does not relax this).

period Named integer vector. Period of each recurrent state; NA for transient states.

is_irreducible Logical. TRUE iff there is exactly one communicating class.

is_aperiodic Logical. TRUE iff every recurrent state has period 1.

is_regular Logical. `is_irreducible && is_aperiodic`.

is_reversible Logical or NA. TRUE iff the chain satisfies detailed balance against its stationary distribution. NA for non-irreducible chains (no unique stationary).

hitting_probabilities $n \times n$ matrix. $[i, j] = P(\text{ever reach } j \text{ starting from } i)$, computed over the same `tol`-thresholded support graph that drives classification so the two are mutually consistent (a state classified "absorbing"/closed never shows hitting probability to states outside its class).

absorption_probabilities $n_{\text{transient}} \times n_{\text{absorbing}}$ matrix or NULL if no transient $\rightarrow$ absorbing pathway exists. $[i, j] = P(\text{eventual absorption in } j \mid \text{start in } i)$.

mean_absorption_time Named numeric vector or NULL. Expected number of steps until absorption from each transient state.

P The (possibly normalized) transition matrix used.

References

Kemeny, J. G. and Snell, J. L. (1976). *Finite Markov Chains*. Springer-Verlag.

Norris, J. R. (1997). *Markov Chains*. Cambridge University Press.

See Also

[passage_time\(\)](#), [markov_stability\(\)](#), [build_network\(\)](#)

Examples

```
net <- build_network(as.data.frame(trjectories), method = "relative")
cs <- chain_structure(net)
print(cs)

summary(cs)
```

chatgpt_srl	<i>ChatGPT Self-Regulated Learning Scale Scores</i>
-------------	---

Description

Scale scores on five self-regulated learning (SRL) constructs for 1,000 responses generated by ChatGPT to a validated SRL questionnaire. Part of a larger dataset comparing LLM-generated responses to human norms across seven large language models.

Usage

chatgpt_srl

Format

- A data frame with 1,000 rows and 5 columns:
- CSU** Numeric. Comprehension and Study Understanding scale mean.
 - IV** Numeric. Intrinsic Value scale mean.
 - SE** Numeric. Self-Efficacy scale mean.
 - SR** Numeric. Self-Regulation scale mean.
 - TA** Numeric. Task Avoidance scale mean.

Source

Vogelsmeier, L.V.D.E., Oliveira, E., Misiejuk, K., Lopez-Pernas, S., & Saqr, M. (2025). Delving into the psychology of Machines: Exploring the structure of self-regulated learning via LLM-generated survey responses. *Computers in Human Behavior*, 173, 108769. doi:10.1016/j.chb.2025.108769

Examples

```
net <- build_network(chatgpt_srl, method = "glasso",
                     params = list(gamma = 0.5))
net
```

clique_expansion *Clique expansion of a hypergraph*

Description

Projects a [net_hypergraph](#) to a standard pairwise [netobject](#) (the *clique expansion* - also called the "downgrade" of a hypergraph to a dyadic graph). Each hyperedge of size k contributes 1 (or its weight) to every pair of its members. The resulting edge weight $W[i, j]$ equals the number of hyperedges containing both i and j (binary incidence) or the sum of incidence products (weighted incidence).

Usage

```
clique_expansion(hg, weighted = TRUE)
```

Arguments

hg	A net_hypergraph object as returned by build_hypergraph() or bipartite_groups() .
weighted	Logical. If TRUE (default), use the hypergraph's incidence values directly (so weighted hypergraphs from bipartite_groups() produce weighted projections). If FALSE, binarise the incidence first so $W[i, j]$ is just the count of shared hyperedges.

Details

The clique expansion is the standard "loss-y but lossless-on-pairwise" projection: it preserves *which pairs co-occurred* and *how often* but discards the higher-order grouping. Comparing `clique_expansion(hg)` to a directly-estimated pairwise network (e.g. via [cooccurrence\(\)](#) on the same data) quantifies how much information was carried by the hyperedge structure.

Computed in one BLAS call via `tcrossprod(incidence)`; runs in $O(n_{\text{nodes}}^2 * n_{\text{hyperedges}})$ time, fast for typical sizes.

Closes the I/O cycle: event data -> [bipartite_groups\(\)](#) -> `clique_expansion()` -> any function that accepts a [netobject](#) (centrality, bootstrap, clustering, plotting via [cograph](#)).

Value

A [netobject](#) (also [cograph_network](#)) with `method = "clique_expansion"`, undirected, with weighted symmetric adjacency $W = \text{incidence} \%*\% \text{t(incidence)}$ and zero diagonal.

Note

(experimental) Validated against `tcrossprod(incidence)` with zero diagonal. No external R package exposes clique expansion as a primitive; the implementation is a direct one-line restatement of the definition.

References

Tian, Y., & Zafarani, R. (2024). Higher-order network analysis methods. *SIGKDD Explorations* 26(1), Section 5.1.5.

See Also

[build_hypergraph\(\)](#), [bipartite_groups\(\)](#), [build_network\(\)](#).

Examples

```
df <- data.frame(
  player = c("A", "B", "C", "A", "B", "D", "C", "D", "E"),
  session = c("S1", "S1", "S1", "S2", "S2", "S3", "S3", "S3", "S3")
)
hg <- bipartite_groups(df, player = "player", group = "session")
net <- clique_expansion(hg)
net$weights
```

cluster_choice

Cluster Choice – sweep k, dissimilarity and method

Description

One-call sweep across any combination of k, dissimilarity metric, and clustering algorithm for distance-based sequence clustering. Mirrors [compare_mmm](#) for model-based clustering: returns a data frame with one row per swept configuration, a best marker on the silhouette-max row in the print method, and a `plot()` that adapts to the swept axes.

Usage

```
cluster_choice(
  data,
  k = 2:5,
  dissimilarity = "hamming",
  method = "ward.D2",
  ...
)
```

Arguments

data	Sequence data (data frame or matrix) – forwarded to build_clusters .
k	Integer vector of cluster counts to sweep. Default 2:5. Each value must be ≥ 2 and $\leq n - 1$.
dissimilarity	Character vector of dissimilarity metrics. Use "all" to expand to every supported metric: <code>c("hamming", "osa", "lv", "dl", "lcs", "qgram", "cosine", "jaccard", "jw")</code> . Default "hamming".

method Character vector of clustering algorithms. Use "all" to expand to every supported method: `c("pam", "ward.D2", "ward.D", "complete", "average", "single", "mcquitty", "median", "centroid")`. Default "ward.D2".

... Other arguments forwarded to `build_clusters` (weighted, lambda, q, p, seed, na_syms, covariates). Note: weighted = TRUE only works with dissimilarity = "hamming" and is rejected up-front when sweeping mixed dissimilarities.

Value

A `cluster_choice` object (a `data.frame` subclass) with one row per (k, dissimilarity, method) combination and columns:

k, dissimilarity, method The configuration for that row.

silhouette Overall average silhouette width (from `cluster::silhouette`, computed inside `build_clusters`).

mean_within_dist Size-weighted mean of within-cluster distances, in the units of the row's dissimilarity.

min_size, max_size, size_ratio Cluster-size balance bounds and their ratio (max / min).

See Also

`build_clusters`, `compare_mmm` for the model-based equivalent, `cluster_diagnostics` for the post-fit diagnostic surface on a single clustering.

Examples

```
seqs <- data.frame(V1 = sample(c("A", "B", "C"), 40, TRUE),
                  V2 = sample(c("A", "B", "C"), 40, TRUE))
cluster_choice(seqs, k = 2:4)

# Sweep dissimilarities at fixed k
cluster_choice(seqs, k = 3, dissimilarity = c("hamming", "lcs", "jaccard"))

# Full grid of k x dissimilarity
cluster_choice(seqs, k = 2:4, dissimilarity = c("hamming", "lcs"))

# "all" sentinel
cluster_choice(seqs, k = 3, dissimilarity = "all")
```

cluster_diagnostics *Cluster Diagnostics*

Description

Unified entry point for clustering quality information. Returns a `net_cluster_diagnostics` object that normalises the diagnostic surface across distance-based and model-based clusterings – you no longer have to know which fields live on `net_clustering` vs. `net_mmm` vs. the slim `net_mmm_clustering` attribute of a `netobject_group`.

Usage

```
cluster_diagnostics(x, ...)

## S3 method for class 'net_cluster_diagnostics'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

x A `net_clustering`, `net_mmm`, `netobject_group` (with `attr(, "clustering")` attached by `cluster_network()` or `cluster_mmm()`), or `net_mmm_clustering`.

... Unsupported. Supplying unused arguments raises an error.

row.names, optional Standard `as.data.frame` arguments (ignored).

Details

The returned object carries:

family Either "distance" or "mmm".

k, n, sizes Number of clusters, number of sequences, sizes vector.

per_cluster A `data.frame` – one row per cluster, columns differ by family. Distance: `cluster`, `size`, `pct`, `mean_within_dist`, `sil_mean`. MMM: `cluster`, `size`, `pct`, `mix_pct`, `avepp`, `class_err_pct`.

overall A named list of family-specific summary metrics (`silhouette` for distance; `avepp_overall`, `entropy`, `classification_error` for MMM).

ics For MMM: a list with BIC, AIC, ICL. NULL for distance.

metadata Method / dissimilarity / weighted / lambda etc.

source The original clustering object, kept by reference so `plot()` can delegate without recomputing anything.

Value

A `net_cluster_diagnostics` object.

See Also

[print.net_cluster_diagnostics](#), [plot.net_cluster_diagnostics](#), [compare_mmm](#) for k-sweep model selection (MMM only).

Examples

```
seqs <- data.frame(V1 = sample(c("A", "B", "C"), 30, TRUE),
                   V2 = sample(c("A", "B", "C"), 30, TRUE))
cl <- build_clusters(seqs, k = 2, method = "ward.D2")
cluster_diagnostics(cl)

grp <- cluster_mmm(seqs, k = 2, n_starts = 1, max_iter = 20, seed = 1)
cluster_diagnostics(grp)
```



```
as.data.frame(cluster_diagnostics(grp))
```

cluster_mmm

Cluster sequences using Mixed Markov Models

Description

Fits a mixture of Markov chains to sequence data and returns a `netobject_group` containing per-cluster transition networks. This is the MMM equivalent of `cluster_network` (which uses distance-based clustering); both functions share the `cluster_by = ...` surface argument so the call shape stays uniform across clustering families.

Usage

```
cluster_mmm(
  data,
  k = 2L,
  n_starts = 50L,
  max_iter = 200L,
  tol = 1e-06,
  smooth = 0.01,
  seed = NULL,
  covariates = NULL,
  covariate_effect = c("em", "posthoc"),
  estimator = c("auto", "firth", "multinom", "chisq"),
  cluster_by = "mmm",
  ...
)
```

Arguments

<code>data</code>	A <code>data.frame</code> (wide format), <code>netobject</code> , or <code>tna</code> model. For <code>tna</code> objects, extracts the stored data.
<code>k</code>	Integer. Whole finite number of mixture components, ≥ 2 . Default: 2.
<code>n_starts</code>	Integer. Positive whole finite number of random restarts. Default: 50.
<code>max_iter</code>	Integer. Positive whole finite maximum EM iterations per start. Default: 200.
<code>tol</code>	Numeric. Finite positive convergence tolerance. Default: $1e-6$.
<code>smooth</code>	Numeric. Finite non-negative Laplace smoothing constant. Default: 0.01.
<code>seed</code>	Integer or <code>NULL</code> . Random seed.
<code>covariates</code>	Optional. Covariates integrated into the EM algorithm to model covariate-dependent mixing proportions. Accepts a string, character vector, formula, or <code>data.frame</code> (same forms as <code>build_clusters</code>). For <code>netobject</code> or <code>cograph_network</code> input, names are resolved against <code>\$metadata</code> first, so a typical call is <code>build_mmm(net, k = 3, covariates = "session_label")</code> . Unlike the post-hoc analysis in <code>build_clusters()</code> , these covariates directly influence cluster membership during EM estimation (see <code>covariate_effect</code>).

covariate_effect	How covariates enter the model. "em" (default) folds them into the EM as covariate-dependent mixing proportions, so they shape the cluster fit itself (and rows with missing covariates are dropped before fitting). "posthoc" fits a plain mixture on every sequence and uses the covariates only for the after-fit multinomial logit, so covariate values — and their missingness — never change which clusters are found. Ignored when covariates is NULL.
estimator	Multinomial fitter for the post-hoc covariate analysis (does not affect EM): "auto" (default) inspects the cluster x covariate cross-tab and falls back to "firth" only when any cell has fewer than 5 observations (separation risk), otherwise the much faster "multinom"; "firth" forces Firth's penalised likelihood via <code>brglm2::brmultinom</code> (finite under separation); "multinom" forces <code>nnet::multinom</code> (warns about separation risk); "chisq" runs descriptive tests (no logit). See build_clusters for full details.
cluster_by	Character. Accepted only as "mmm" (the default). Present so <code>cluster_mmm()</code> and <code>cluster_network()</code> share the same call shape; any other value raises an error pointing at cluster_network .
...	Unsupported. Supplying unused arguments raises an error.

Details

For the full `net_mmm` object with posterior probabilities, model fit statistics, and S3 methods, use [build_mmm](#) instead.

Value

A `netobject_group` (list of `netobjects`, one per cluster). MMM-specific information is stored in `attr(, "clustering")` (class `"net_mmm_clustering"`):

assignments Integer vector of cluster assignments.

k Number of clusters.

posterior N x k matrix of posterior probabilities.

mixing Mixing proportions.

quality List with AvePP, entropy, classification error.

BIC, AIC, ICL Model fit statistics.

data The full N-row sequence frame, matching `$assignments` – so [sequence_plot](#) and [distribution_plot](#) can recover both.

See Also

[build_mmm](#) for the full MMM object, [cluster_network](#) for distance-based clustering

Examples

```
seqs <- data.frame(V1 = sample(c("A", "B", "C"), 30, TRUE),
                  V2 = sample(c("A", "B", "C"), 30, TRUE))
grp <- cluster_mmm(seqs, k = 2, n_starts = 1, max_iter = 10, seed = 1)
```

```

grp[[1]]$weights
attr(grp, "clustering")$assignments

# Visualise with sequence_plot
seqs <- data.frame(
  V1 = sample(LETTERS[1:3], 40, TRUE),
  V2 = sample(LETTERS[1:3], 40, TRUE),
  V3 = sample(LETTERS[1:3], 40, TRUE)
)
grp <- cluster_mmm(seqs, k = 2)
sequence_plot(grp, type = "index")

```

cluster_network	<i>Cluster data and build per-cluster networks in one step</i>
-----------------	--

Description

Combines sequence clustering and network estimation into a single call. Clusters the data using the specified algorithm, then calls [build_network](#) on each cluster subset.

Usage

```
cluster_network(data, k, cluster_by = "pam", dissimilarity = "hamming", ...)
```

Arguments

data	Sequence data. Accepts a data frame, matrix, or netobject. See build_clusters for supported formats.
k	Integer. Number of clusters.
cluster_by	Character. Clustering algorithm passed to build_clusters 's method parameter ("pam", "ward.D2", "ward.D", "complete", "average", "single", "mcquitty", "median", "centroid"), or "mmm" for Mixed Markov Model clustering. Default: "pam".
dissimilarity	Character. Distance metric for sequence clustering. Only valid when cluster_by != "mmm". Default: "hamming".
...	Routed to two stages. For distance clustering (cluster_by != "mmm"), build_clusters arguments na_syms, weighted, lambda, seed, q, p, and covariates are intercepted and forwarded to the clusterer; recognised network-estimation arguments flow to build_network . Unknown arguments error before either stage is run. When cluster_by = "mmm", the recognised build_mmm arguments (n_starts, max_iter, tol, smooth, seed, covariates) are intercepted instead, and the rest flows to build_network. In both modes, when data is a netobject, its build_args are merged into the build_network side (caller's explicit values take precedence).

Details

If data is a netobject and method is not provided in . . . , the original network method is inherited automatically so the per-cluster networks match the type of the input network.

Value

A netobject_group.

See Also

[build_clusters](#), [cluster_mmm](#), [build_network](#)

Examples

```
seqs <- data.frame(V1 = c("A", "B", "C", "A", "B"), V2 = c("B", "C", "A", "B", "A"),
                  V3 = c("C", "A", "B", "C", "B"))
grp <- cluster_network(seqs, k = 2)
grp

set.seed(1)
seqs <- data.frame(
  V1 = sample(LETTERS[1:4], 50, TRUE), V2 = sample(LETTERS[1:4], 50, TRUE),
  V3 = sample(LETTERS[1:4], 50, TRUE), V4 = sample(LETTERS[1:4], 50, TRUE)
)
# Default: PAM clustering, relative (transition) networks
grp <- cluster_network(seqs, k = 3)

# Specify network method (cor requires numeric panel data)
## Not run:
panel <- as.data.frame(matrix(rnorm(1500), nrow = 300, ncol = 5))
grp <- cluster_network(panel, k = 2, method = "cor")

## End(Not run)

# MMM-based clustering
grp <- cluster_network(seqs, k = 2, cluster_by = "mmm")
```

Description

Aggregates node-level network weights to cluster-level summaries. Computes both between-cluster transitions (how clusters connect to each other) and within-cluster transitions (how nodes connect within each cluster).

Usage

```
cluster_summary(
  x,
  clusters = NULL,
  method = c("sum", "mean", "median", "max", "min", "density", "geomean"),
  directed = TRUE,
  compute_within = TRUE
)
```

Arguments

- x** Network input. Accepts multiple formats:
- matrix** Numeric adjacency/weight matrix. Row and column names are used as node labels. Values represent edge weights (e.g., transition counts, co-occurrence frequencies, or probabilities).
 - netobject** A cograph network object. The function extracts the weight matrix from `x$weights` or converts via `to_matrix()`. Clusters can be auto-detected from node attributes.
 - tna** A tna object from the tna package. Extracts `x$weights`.
 - cluster_summary** If already a cluster_summary, returns unchanged.
- clusters** Cluster/group assignments for nodes. Accepts multiple formats:
- NULL** (default) Auto-detect from netobject. Looks for columns named 'clusters', 'cluster', 'groups', or 'group' in `x$nodes`. Throws an error if no cluster column is found. This option only works when `x` is a netobject.
 - vector** Cluster membership for each node, in the same order as the matrix rows/columns. Can be numeric (1, 2, 3) or character ("A", "B"). Cluster names will be derived from unique values. Example: `c(1, 1, 2, 2, 3, 3)` assigns first two nodes to cluster 1.
 - data.frame** A data frame where the first column contains node names and the second column contains group/cluster names. Example: `data.frame(node = c("A", "B", "C"), group = c("G1", "G1", "G2"))`
 - named list** Explicit mapping of cluster names to node labels. List names become cluster names, values are character vectors of node labels that must match matrix row/column names. Example: `list(Alpha = c("A", "B"), Beta = c("C", "D"))`
- method** Aggregation method for combining edge weights within/between clusters. Controls how multiple node-to-node edges are summarized:
- "sum"** (default) Sum of all edge weights. Best for count data (e.g., transition frequencies). Preserves total flow.
 - "mean"** Average edge weight. Best when cluster sizes differ and you want to control for size. Note: when input is already a transition matrix (rows sum to 1), "mean" avoids size bias. Example: cluster with 5 nodes won't have 5x the weight of cluster with 1 node.
 - "median"** Median edge weight. Robust to outliers.
 - "max"** Maximum edge weight. Captures strongest connection.

	"min" Minimum edge weight. Captures weakest connection.
	"density" Sum of (non-zero) edge weights divided by the number of possible edges between the two clusters ($n_i * n_j$). Normalizes by cluster size combinations. Because zero/NA edges are stripped before aggregation, this equals "mean" exactly when the cluster-pair block is fully dense (no zero edges), and is strictly smaller than "mean" when zero edges are present (it divides by the larger possible-edge count).
	"geomean" Geometric mean of positive weights. Useful for multiplicative processes.
directed	Logical. If TRUE (default), treat network as directed. A->B and B->A are separate edges. If FALSE, edges are undirected and the matrix is symmetrized before processing.
compute_within	Logical. If TRUE (default), compute within-cluster transition matrices for each cluster. Each cluster gets its own $n_i \times n_i$ matrix showing internal node-to-node transitions. Set to FALSE to skip this computation for better performance when only between-cluster summary is needed.

Details

This is the core function for Multi-Cluster Multi-Level (MCML) analysis. Use `as_tna()` to convert results to tna objects for further analysis with the tna package.

Workflow:

Typical MCML analysis workflow:

```
# 1. Create network
net <- build_network(data, method = "relative")
net$nodes$clusters <- group_assignments

# 2. Compute cluster summary (arithmetic aggregation over edges)
cs <- cluster_summary(net, method = "sum")

# 3. Convert to tna models (normalization happens in as_tna)
tna_models <- as_tna(cs)

# 4. Analyze/visualize
plot(tna_models$macro)
tna::centralities(tna_models$macro)
```

Between-Cluster Matrix Structure:

The `macro$weights` matrix has clusters as both rows and columns:

- Off-diagonal (row i , col j): Aggregated weight from cluster i to cluster j
- Diagonal (row i , col i): Within-cluster total (aggregation of internal edges)

Rows are NOT normalized. Entries are elementwise aggregates produced by method. If the caller wants probabilities, they should normalize downstream (e.g. via `as_tna()`). Mixing an arithmetic aggregation with row-normalization here (the old type = "tna" combined with `method = "min"` / "mean" etc.) produces numbers that sum to 1 per row but are not a probability distribution over any process; that silently-wrong combination is why type was removed from the matrix path. The

sequence and edgelist paths of `build_mcml()` keep type, where the aggregation is always counts and the post-processing chooses between well-defined network constructions.

Choosing method:

Input data	Recommended method	Reason
Edge counts	"sum"	Preserves total flow between clusters
Transition matrix	"mean"	Avoids cluster size bias
Correlation matrix	"mean"	Average correlations
Dense weighted	"max" / "median"	Robust summary

Value

A `cluster_summary` object (S3 class) containing:

between List with two elements:

weights $k \times k$ matrix of cluster-to-cluster weights, where k is the number of clusters. Row i , column j contains the elementwise aggregation (per method) of all edges from nodes in cluster i to nodes in cluster j . Diagonal contains within-cluster totals. Pure arithmetic – no row normalization.

inits Numeric vector of length k . Initial state distribution across clusters, computed from column sums of the original matrix. Represents the proportion of incoming edges to each cluster.

within Named list with one element per cluster. Each element contains:

weights $n_i \times n_i$ matrix for nodes within that cluster. Shows internal transitions between nodes in the same cluster.

inits Initial distribution within the cluster.

NULL if `compute_within = FALSE`.

clusters Named list mapping cluster names to their member node labels. Example: `list(A = c("n1", "n2"), B = c("n3", "n4", "n5"))`

meta List of metadata:

method The method argument used ("sum", "mean", etc.)

directed Logical, whether network was treated as directed

n_nodes Total number of nodes in original network

n_clusters Number of clusters

cluster_sizes Named vector of cluster sizes

See Also

`as_tna()` to convert results to tna objects, `plot()` for two-layer visualization, `plot()` for flat cluster visualization

Examples

```
# -----
# Basic usage with matrix and cluster vector
# -----
mat <- matrix(runif(100), 10, 10)
rownames(mat) <- colnames(mat) <- LETTERS[1:10]

clusters <- c(1, 1, 1, 2, 2, 2, 3, 3, 3, 3)
cs <- cluster_summary(mat, clusters)

# Access results
cs$macro$weights      # 3x3 cluster transition matrix
cs$macro$inits        # Initial distribution
cs$clusters$1`$weights # Within-cluster 1 transitions
cs$meta              # Metadata

# -----
# Named list clusters (more readable)
# -----
clusters <- list(
  Alpha = c("A", "B", "C"),
  Beta  = c("D", "E", "F"),
  Gamma = c("G", "H", "I", "J")
)
cs <- cluster_summary(mat, clusters)
cs$macro$weights      # Rows/cols named Alpha, Beta, Gamma
cs$clusters$Alpha     # Within Alpha cluster

# -----
# Auto-detect clusters from netobject
# -----

seqs <- data.frame(
  V1 = sample(LETTERS[1:10], 30, TRUE), V2 = sample(LETTERS[1:10], 30, TRUE),
  V3 = sample(LETTERS[1:10], 30, TRUE)
)
net <- build_network(seqs, method = "relative")
cs2 <- cluster_summary(net, c(1, 1, 1, 2, 2, 2, 3, 3, 3, 3))

# -----
# Different aggregation methods
# -----
cs_sum <- cluster_summary(mat, clusters, method = "sum") # Total flow
cs_mean <- cluster_summary(mat, clusters, method = "mean") # Average
cs_max <- cluster_summary(mat, clusters, method = "max") # Strongest

# -----
# Skip within-cluster computation for speed
# -----
cs_fast <- cluster_summary(mat, clusters, compute_within = FALSE)
cs_fast$clusters # NULL
```



```
# -----
# Convert to tna objects for tna package
# (as_tna() applies its own row normalisation)
# -----
cs <- cluster_summary(mat, clusters, method = "sum")
tna_models <- as_tna(cs)
# tna_models$macro      # tna object
# tna_models$clusters$Alpha # tna object
```

coefs

*Tidy coefficients from a fitted mlvar model***Description**

Generic accessor for the tidy coefficient table stored on a `build_mlvar()` result. Returns a `data.frame` with one row per (outcome, predictor) pair and columns outcome, predictor, beta, se, t, p, ci_lower, ci_upper, significant.

Usage

```
coefs(x, ...)
```

```
## S3 method for class 'net_mlvar'
coefs(x, ...)
```

```
## Default S3 method:
coefs(x, ...)
```

Arguments

```
x          A fitted model object - currently only net_mlvar is supported.
...        Unused.
```

Details

Only the within-person (temporal) coefficients are tabulated - these are the lagged fixed effects that populate `fit$temporal`. The between-subjects effects that go into `fit$between` are handled via the D (I - Gamma) transformation and are not exposed as a separate tidy table.

Value

A tidy `data.frame` of coefficient estimates.

Examples

```
## Not run:
d <- simulate_data("mlvar", seed = 1)
fit <- build_mlvar(d, vars = attr(d, "vars"),
                  id = "id", day = "day", beep = "beep")

print(fit)
summary(fit)

## End(Not run)
```

compare_mmm

*Compare MMM fits across different k***Description**

Compare MMM fits across different k

Usage

```
compare_mmm(data, k = 2:5, return_fits = FALSE, ...)
```

Arguments

data	Data frame, netobject, or tna model.
k	Integer vector of component counts. Values must be whole finite numbers ≥ 2 . Default: 2:5.
return_fits	Logical. When TRUE the fitted models are retained on the result via <code>attr(result, "fits")</code> (a list of <code>net_mmm</code> objects, named by k), so the user can pick the chosen model without re-running the EM. Default FALSE keeps the historical lightweight return shape – only the comparison table is allocated.
...	Arguments passed to <code>build_mmm</code> .

Value

A `mmm_compare` data frame with BIC, AIC, ICL, AvePP, entropy per k. When `return_fits = TRUE`, the fitted models are attached as `attr(result, "fits")`.

Examples

```
seqs <- data.frame(V1 = sample(c("A", "B", "C"), 30, TRUE),
                  V2 = sample(c("A", "B", "C"), 30, TRUE))
comp <- compare_mmm(seqs, k = 2:3, n_starts = 1, max_iter = 10, seed = 1)
comp

seqs <- data.frame(
  V1 = sample(LETTERS[1:3], 30, TRUE), V2 = sample(LETTERS[1:3], 30, TRUE),
```

```

V3 = sample(LETTERS[1:3], 30, TRUE), V4 = sample(LETTERS[1:3], 30, TRUE)
)
comp <- compare_mmm(seqs, k = 2:3, seed = 42)
print(comp)

# Retain fits to avoid a re-fit after picking the BIC-min model.
comp_with_fits <- compare_mmm(seqs, k = 2:3, seed = 42, return_fits = TRUE)
best_k <- comp_with_fits$`k`[which.min(comp_with_fits$BIC)]
best_fit <- attr(comp_with_fits, "fits")[[as.character(best_k)]]

```

compare_model

Compare two networks descriptively

Description

Computes a battery of descriptive comparison metrics between two networks or two weight matrices: weight deviations (mean / median / RMS / max absolute difference, relative mean absolute difference, coefficient-of- variation ratio), four correlation measures (Pearson, Spearman, Kendall, distance correlation), five dissimilarity measures (Euclidean, Manhattan, Canberra, Bray-Curtis, Frobenius), five similarity measures (Cosine, Jaccard, Dice, Overlap, RV), pattern agreements, and side-by-side network metrics. Optionally adds centrality differences and centrality correlations.

Usage

```

compare_model(x, ...)

## S3 method for class 'netobject'
compare_model(
  x,
  y,
  scaling = "none",
  measures = character(0),
  network = TRUE,
  ...
)

## S3 method for class 'cograph_network'
compare_model(
  x,
  y,
  scaling = "none",
  measures = character(0),
  network = TRUE,
  ...
)

```

```
## S3 method for class 'matrix'
compare_model(
  x,
  y,
  scaling = "none",
  measures = character(0),
  network = TRUE,
  ...
)
```

Arguments

x	A netobject, cograph_network, or numeric square matrix.
...	Ignored.
y	A netobject, cograph_network, or numeric square matrix.
scaling	Scaling applied to both weight matrices before comparison. One of: "none" Identity (default). "minmax" $(w - \min)/(\max - \min)$; maps to $[0, 1]$. "max" $w / \max(w)$; preserves sign. "rank" Min-max of average ranks; ordinal scaling. "zscore" $(w - \bar{w})/s_w$; standard score. "robust" $(w - \text{med}(w))/\text{mad}(w)$; Huber-style robust z-score, resists outliers. "log" $\log(w)$; requires $w > 0$. "log1p" $\log(1 + w)$; admits $w \geq 0$. "softmax" Numerically stable softmax over the flattened vector. "quantile" Empirical CDF of the flattened vector. "frobenius" Divide the matrix by its Frobenius norm $\ W\ _F = \sqrt{\sum w_{ij}^2}$; matrix-level normalisation. "row" Row-stochastic normalisation (each row's absolute values sum to 1). Only meaningful for non-negative matrices; rows summing to zero are left unchanged. Scalings that produce negative weights (zscore, robust) are compatible with network = TRUE because the side-by-side metrics use Nestimate's base-R Floyd-Warshall, which handles negative weights.
measures	Character vector of centrality measures to compare. Empty by default (no centrality block). Valid names are "InStrength", "OutStrength", and "Betweenness". Unknown names are ignored with a warning.
network	Logical. Include side-by-side network metrics from summary()? Default TRUE.

Details

Mirrors `tna::compare()` numerically. Inputs are converted to weight matrices and scaled before comparison; the choice of scaling determines how weights from different estimators are placed on a common footing.

Value

A net_comparison object: a named list with matrices, difference_matrix, edge_metrics, summary_metrics, optionally network_metrics, centrality_differences, centrality_correlations.

compare_model.netobject_group

Compare two networks within a netobject_group

Description

Selects two members of a netobject_group (by index or name) and dispatches to compare_model.netobject().

Usage

```
## S3 method for class 'netobject_group'
compare_model(
  x,
  i = 1L,
  j = 2L,
  scaling = "none",
  measures = character(0),
  network = TRUE,
  ...
)
```

Arguments

x	A netobject_group.
i	Integer or character. Index or name of the first network. Default 1L.
j	Integer or character. Index or name of the second network. Default 2L.
scaling	See compare_model().
measures	See compare_model().
network	See compare_model().
...	Passed to compare_model.netobject().

Value

A net_comparison object.

convert_sequence_format

Convert Sequence Data to Different Formats

Description

Convert wide or long sequence data into frequency counts, one-hot encoding, edge lists, or follows format.

Usage

```
convert_sequence_format(
  data,
  seq_cols = NULL,
  id_col = NULL,
  action = NULL,
  time = NULL,
  format = c("frequency", "onehot", "edgelist", "follows")
)
```

Arguments

data	Data frame containing sequence data.
seq_cols	Character vector. Names of columns containing sequential states (for wide format input). If NULL, all columns except id_col are used. Default: NULL.
id_col	Character vector. Name(s) of the ID column(s). For long format, required. For wide format, optional: if NULL, the first column is used as the id only when it is a genuine identifier (its values are disjoint from the states in the remaining columns); for canonical wide sequence data with no id column (e.g. V1..Vn / T1..Tn), a row-index id is synthesized and every column is treated as a sequence column. Default: NULL.
action	Character or NULL. Name of the column containing actions/states (for long format input). If provided, data is treated as long format. Default: NULL.
time	Character or NULL. Name of the time column for ordering actions within sequences (for long format). Default: NULL.
format	Character. Output format: "frequency" Count of each action per sequence (wide, one column per state). "onehot" Binary presence/absence of each action per sequence. "edgelist" Consecutive transition pairs (from, to) per sequence. "follows" Each action paired with the action that preceded it.

Value

A data frame in the requested format:

frequency ID columns + one integer column per state with counts.

onehot ID columns + one binary column per state (0/1).

edgelist ID columns + from and to columns.

follows ID columns + act and follows columns.

See Also

[frequencies](#) for building transition frequency matrices.

Examples

```
# Wide format input
seqs <- data.frame(V1 = c("A", "B", "A"), V2 = c("B", "A", "C"), V3 = c("A", "C", "B"))
convert_sequence_format(seqs, format = "frequency")
convert_sequence_format(seqs, format = "edgelist")
```

cooccurrence

Build a Co-occurrence Network

Description

Constructs an undirected co-occurrence network from various input formats. Entities that appear together in the same transaction, document, or record are connected, with edge weights reflecting raw counts or a similarity measure. Argument names follow the citenets convention.

Usage

```
cooccurrence(
  data,
  field = NULL,
  by = NULL,
  sep = NULL,
  similarity = c("none", "jaccard", "cosine", "inclusion", "association", "dice",
    "equivalence", "relative"),
  threshold = 0,
  min_occur = 1L,
  diagonal = TRUE,
  top_n = NULL,
  ...
)
```

Arguments

data	Input data. Accepts: • A data.frame with a delimited column (field + sep). • A data.frame in long/bipartite format (field + by). • A binary (0/1) data.frame or matrix (auto-detected). • A wide sequence data.frame or matrix (non-binary). • A list of character vectors (each element is a transaction).
field	Character. The entity column - determines what the nodes are. For delimited format, a single column whose values are split by sep. For long/bipartite format, the item column. For multi-column delimited, a vector of column names whose split values are pooled per row.
by	Character or NULL. What links the nodes. For long/bipartite format, the grouping column (e.g., "paper_id", "session_id"). Each unique value of by defines one transaction. If NULL (default), entities co-occur within the same row/document.
sep	Character or NULL. Separator for splitting delimited fields (e.g., ";", ",", "). Default NULL.
similarity	Character. Similarity measure applied to the raw co-occurrence counts. One of: "none" Raw co-occurrence counts. "jaccard" $C_{ij}/(f_i + f_j - C_{ij})$. "cosine" $C_{ij}/\sqrt{f_i \cdot f_j}$ (Salton's cosine). "inclusion" $C_{ij}/\min(f_i, f_j)$ (Simpson coefficient). "association" $C_{ij}/(f_i \cdot f_j)$ (association strength / probabilistic affinity index; van Eck & Waltman, 2009). "dice" $2C_{ij}/(f_i + f_j)$. "equivalence" $C_{ij}^2/(f_i \cdot f_j)$ (Salton's cosine squared). "relative" Row-normalized: each row sums to 1.
threshold	Numeric. Minimum edge weight to retain. Edges below this value are set to zero. Applied <i>after</i> similarity normalization. Default 0.
min_occur	Integer. Minimum entity frequency (number of transactions an entity must appear in). Entities below this threshold are dropped before computing co-occurrence. Default 1 (keep all).
diagonal	Logical. If TRUE (default), the diagonal of the co-occurrence matrix is kept (item self-co-occurrence = item frequency). If FALSE, the diagonal is zeroed.
top_n	Integer or NULL. If specified, return only the top top_n edges by weight. Default NULL (all edges).
...	Currently unused.

Details

Six input formats are supported, auto-detected from the combination of field, by, and sep:

1. **Delimited:** field + sep (single column). Each cell is split by sep, trimmed, and de-duplicated per row.

2. **Multi-column delimited:** field (vector) + sep. Values from multiple columns are split, pooled, and de-duplicated per row.
3. **Long bipartite:** field + by. Groups by by; unique values of field within each group form a transaction.
4. **Binary matrix:** No field/by/sep, all values 0/1. Columns are items, rows are transactions.
5. **Wide sequence:** No field/by/sep, non-binary. Unique values across each row form a transaction.
6. **List:** A plain list of character vectors.

The pipeline converts all formats into a list of character vectors (transactions), optionally filters by `min_occur`, builds a binary transaction matrix, computes `crossprod(B)` for the raw co-occurrence counts, normalizes via the chosen similarity, then applies threshold and `top_n` filtering.

Value

A netobject (undirected) with `method = "co_occurrence_fn"`. The `$weights` matrix contains similarity (or raw) co-occurrence values. The `$params` list stores the similarity method, threshold, and the number of transactions.

References

van Eck, N. J., & Waltman, L. (2009). How to normalize co-occurrence data? An analysis of some well-known similarity measures. *Journal of the American Society for Information Science and Technology*, 60(8), 1635–1651.

See Also

[build_cna](#) for sequence-positional co-occurrence via `build_network()`.

Examples

```
# Delimited field (e.g., keyword co-occurrence)
df <- data.frame(
  id = 1:4,
  keywords = c("network; graph", "graph; matrix; network",
               "matrix; algebra", "network; algebra; graph")
)
net <- cooccurrence(df, field = "keywords", sep = ";")

# Long/bipartite
long_df <- data.frame(
  paper = c(1, 1, 1, 2, 2, 3, 3),
  keyword = c("network", "graph", "matrix", "graph", "algebra",
              "network", "algebra")
)
net <- cooccurrence(long_df, field = "keyword", by = "paper")

# List of transactions
transactions <- list(c("A", "B"), c("B", "C"), c("A", "B", "C"))
net <- cooccurrence(transactions, similarity = "jaccard")
```

```
# Binary matrix
bin <- matrix(c(1,0,1, 1,1,0, 0,1,1), nrow = 3, byrow = TRUE,
              dimnames = list(NULL, c("X", "Y", "Z")))
net <- cooccurrence(bin)
```

distribution_plot	<i>State Distribution Plot Over Time</i>
-------------------	--

Description

Draws how state proportions (or counts) evolve across time points. For each time column, tabulates how many sequences are in each state and renders the result as a stacked area (default) or stacked bar chart. Accepts the same inputs as [sequence_plot](#).

Usage

```
distribution_plot(
  x,
  group = NULL,
  scale = c("proportion", "count"),
  geom = c("area", "bar"),
  na = TRUE,
  trim = NULL,
  trim_clusterwise = FALSE,
  state_colors = NULL,
  na_color = "grey90",
  frame = FALSE,
  width = NULL,
  height = NULL,
  main = NULL,
  show_n = TRUE,
  time_label = "Time",
  xlab = NULL,
  y_label = NULL,
  ylab = NULL,
  tick = NULL,
  ncol = NULL,
  nrow = NULL,
  combined = TRUE,
  legend = c("right", "bottom", "none"),
  legend_size = NULL,
  legend_title = NULL,
  legend_ncol = NULL,
  legend_border = NA,
  legend_bty = "n"
)
```

Arguments

x	Wide-format sequence data. Accepts the same inputs as sequence_plot : <code>data.frame</code> , <code>matrix</code> , <code>netobject</code> , <code>net_clustering</code> , <code>netobject_group</code> , <code>net_mmm</code> , or <code>tna</code> . When clustering info is available, one panel is drawn per cluster.
group	Optional grouping vector (length <code>nrow(x)</code>) producing one panel per group. Ignored if <code>x</code> is a <code>net_clustering</code> .
scale	"proportion" (default) divides each column by its total so bands fill 0..1. "count" keeps raw counts.
geom	"area" (default) draws stacked polygons; "bar" draws stacked bars.
na	If TRUE (default), NA cells are shown as an extra band coloured <code>na_color</code> .
trim	Optional time-axis truncation, to stop a few long sequences from stretching the plot. NULL (default) keeps the full width. A fraction in (0, 1) drops everything past that quantile of sequence lengths (e.g. <code>trim = 0.95</code>); a value ≥ 1 is an absolute cut (<code>trim = 50</code> keeps the first 50 time points). See sequence_plot .
trim_clusterwise	Grouped plots only, fractional trim only. FALSE (default) uses one pooled cutoff for every panel so the time axes stay aligned; TRUE crops each group to its own length quantile (panels can differ in width). See sequence_plot .
state_colors	Vector of colours, one per state. Defaults to Okabe-Ito.
na_color	Colour for the NA band.
frame	If TRUE (default), draw a box around each panel.
width, height	Optional device dimensions. See sequence_plot .
main	Plot title.
show_n	Append "(n = N)" (per-group when grouped) to the title.
time_label	X-axis label.
xlab	Alias for <code>time_label</code> .
y_label	Y-axis label. Defaults to "Proportion" or "Count" based on scale.
ylab	Alias for <code>y_label</code> .
tick	Show every Nth x-axis label. NULL = auto.
ncol, nrow	Facet grid dimensions. NULL = auto: <code>ncol = ceiling(sqrt(G))</code> , <code>nrow = ceiling(G / ncol)</code> . Ignored when <code>combined = FALSE</code> .
combined	When TRUE (default), groups are arranged on one figure via <code>graphics::layout()</code> . When FALSE, each group is drawn on its own page (one full-size figure per group, with its own legend). Useful when you want each group at full size in knitr (<code>fig.show = "asis"</code>) or to save each as a separate file. Single-group calls (<code>G == 1</code>) ignore this argument.
legend	Legend position: "right" (default), "bottom", or "none".
legend_size	Legend text size. NULL (default) auto-scales from device width (clamped to [0.65, 1.2]).
legend_title	Optional legend title.
legend_ncol	Number of legend columns.
legend_border	Swatch border colour.
legend_bty	"n" (borderless) or "o" (boxed).

Value

Invisibly, a list with counts, proportions, levels, palette, and groups.

See Also

[sequence_plot](#), [build_clusters](#)

Examples

```
distribution_plot(as.data.frame(trjectories))
```

estimate_network	<i>Estimate a Network (Deprecated)</i>
------------------	--

Description

This function is deprecated. Use [build_network](#) instead.

Usage

```
estimate_network(
  data,
  method = "relative",
  params = list(),
  scaling = NULL,
  threshold = 0,
  level = NULL,
  ...
)
```

Arguments

data	Data frame (sequences or per-observation frequencies) or a square symmetric matrix (correlation or covariance).
method	Character. Defaults to "relative" for backward compatibility.
params	Named list. Method-specific parameters passed to the estimator function (e.g. <code>list(gamma = 0.5)</code> for <code>glasso</code> , or <code>list(format = "wide")</code> for transition methods). This is the key composability feature: downstream functions like <code>bootstrap</code> or <code>grid search</code> can store and replay the full <code>params</code> list without knowing method internals. Transition estimators accept <code>tna</code> -style sequence options such as <code>weighted</code> and <code>concat</code> (and the low-level <code>begin_state</code> / <code>end_state</code> , of which <code>start</code> / <code>end</code> are the public form – see those arguments). Column-like entries in <code>params</code> (<code>action</code> , <code>id</code> , <code>id_col</code> , <code>time</code> , <code>session</code> , <code>order</code> , <code>codes</code> , and <code>group</code>) are resolved before format detection and must name existing columns. If the same column role is supplied both directly and through <code>params</code> , the names must agree.

scaling	Character vector or NULL. Post-estimation scaling to apply (in order). Options: "minmax", "max", "rank", "normalize". Can combine: c("rank", "minmax"). Default: NULL (no scaling).
threshold	Numeric. Absolute values below this are set to zero in the result matrix. Default: 0 (no thresholding).
level	Character or NULL. Multilevel decomposition for association methods. One of NULL, "between", "within", "both". Requires id_col. Default: NULL.
...	Additional arguments passed to build_network .

Value

A netobject (see [build_network](#)).

See Also

[build_network](#)

Examples

```
data <- data.frame(A = c("x", "y", "z", "x"), B = c("y", "x", "z", "y"))
net <- estimate_network(data, method = "relative")
```

euler_characteristic *Euler Characteristic*

Description

Computes $\chi = \sum_{k=0}^d (-1)^k f_k$ where f_k is the number of k-simplices. By the Euler-Poincare theorem, $\chi = \sum_k (-1)^k \beta_k$.

Usage

```
euler_characteristic(sc)
```

Arguments

sc A simplicial_complex object.

Value

Integer.

Examples

```
mat <- matrix(c(0,.6,.5,.6,0,.4,.5,.4,0), 3, 3)
colnames(mat) <- rownames(mat) <- c("A","B","C")
sc <- build_simplicial(mat, threshold = 0.3)
euler_characteristic(sc)
```

evaluate_links	<i>Evaluate Link Predictions Against Known Edges</i>
----------------	--

Description

Computes AUC-ROC, precision@k, and average precision for link predictions against a set of known true edges.

Usage

```
evaluate_links(pred, true_edges, k = c(5L, 10L, 20L))
```

Arguments

pred	A net_link_prediction object.
true_edges	A data frame with columns from and to, or a binary matrix where 1 indicates a true edge.
k	Integer vector. Values of k for precision@k. Default: c(5, 10, 20).

Value

A data frame with columns: method, auc, average_precision, and one precision_at_k column per k value.

Examples

```
set.seed(42)
seqs <- data.frame(
  V1 = sample(LETTERS[1:5], 50, TRUE),
  V2 = sample(LETTERS[1:5], 50, TRUE),
  V3 = sample(LETTERS[1:5], 50, TRUE)
)
net <- build_network(seqs, method = "relative")
pred <- predict_links(net, exclude_existing = FALSE)

# Evaluate: predict the network's own edges
true <- data.frame(from = pred$predictions$from[1:5],
  to = pred$predictions$to[1:5])
evaluate_links(pred, true)
```

extract_edges	<i>Extract Edge List with Weights</i>
---------------	---------------------------------------

Description

Extract an edge list from a TNA model, representing the network as a data frame of from-to-weight tuples.

Usage

```
extract_edges(model, threshold = 0, include_self = FALSE, sort_by = "weight")
```

Arguments

model	A TNA model object or a matrix of weights.
threshold	Numeric. Minimum weight to include an edge. Default: 0.
include_self	Logical. Whether to include self-loops. Default: FALSE.
sort_by	Character. Column to sort by: "weight" (descending), "from", "to", or NULL for no sorting. Default: "weight".

Details

This function converts the transition matrix into an edge list format, which is useful for visualization, analysis with igraph, or export to other network tools.

Value

A data frame with columns:

from Source state name.

to Target state name.

weight Edge weight (transition probability).

See Also

[extract_transition_matrix](#) for the full matrix, [build_network](#) for network estimation.

Examples

```
seqs <- data.frame(V1 = c("A", "B", "A"), V2 = c("B", "A", "C"), V3 = c("A", "C", "B"))
net <- build_network(seqs, method = "relative")
edges <- extract_edges(net, threshold = 0.05)
head(edges)
```

extract_initial_probs *Extract Initial Probabilities from Model*

Description

Extract the initial state probability vector from a TNA model object.

Usage

```
extract_initial_probs(model)
```

Arguments

model A TNA model object or a list containing an 'initial' element.

Details

Initial probabilities represent the probability of starting a sequence in each state. If the model doesn't have explicit initial probabilities, this function attempts to estimate them from the data or use uniform probabilities.

Value

A named numeric vector of initial state probabilities.

See Also

[extract_transition_matrix](#) for extracting the transition matrix, [extract_edges](#) for extracting an edge list.

Examples

```
seqs <- data.frame(V1 = c("A", "B", "A"), V2 = c("B", "A", "C"), V3 = c("A", "C", "B"))
net <- build_network(seqs, method = "relative")
init_probs <- extract_initial_probs(net)
print(init_probs)
```

`extract_transition_matrix`*Extract Transition Matrix from Model*

Description

Extract the transition probability matrix from a TNA model object.

Usage

```
extract_transition_matrix(model, type = c("raw", "scaled"))
```

Arguments

<code>model</code>	A TNA model object or a list containing a 'weights' element.
<code>type</code>	Character. Type of matrix to return: "raw" The raw weight matrix as stored in the model. "scaled" Row-normalized to ensure rows sum to 1. Default: "raw".

Details

TNA models store transition weights in different locations depending on the model type. This function handles the extraction automatically.

For "scaled" type, each row is divided by its sum to create valid transition probabilities. This is useful when the original weights don't sum to 1.

Value

A square numeric matrix with row and column names as state names.

See Also

[extract_initial_probs](#) for extracting initial probabilities, [extract_edges](#) for extracting an edge list.

Examples

```
seqs <- data.frame(V1 = c("A", "B", "A"), V2 = c("B", "A", "C"), V3 = c("A", "C", "B"))
net <- build_network(seqs, method = "relative")
trans_mat <- extract_transition_matrix(net)
print(trans_mat)
```

frequencies

*Sequence Data Conversion Functions***Description**

Functions for converting sequence data (long or wide format) into transition frequency matrices and other useful representations.

Convert long or wide format sequence data into a transition frequency matrix. Counts how many times each transition from state_i to state_j occurs across all sequences.

Usage

```
frequencies(
  data,
  action = "Action",
  id = NULL,
  time = "Time",
  cols = NULL,
  format = c("auto", "long", "wide")
)
```

Arguments

data	Data frame containing sequence data in long or wide format.
action	Character. Name of the column containing actions/states (for long format). Default: "Action".
id	Character vector. Name(s) of the column(s) identifying sequences. For long format, each unique combination of ID values defines a sequence. For wide format, used to exclude non-state columns. Default: NULL.
time	Character. Name of the time column used to order actions within sequences (for long format). Default: "Time".
cols	Character vector. Names of columns containing states (for wide format). If NULL, all non-ID columns are used. Default: NULL.
format	Character. Format of input data: "auto" (detect automatically), "long", or "wide". Default: "auto".

Details

For **long format** data, each row is a single action/event. Sequences are defined by the `id` column(s), and actions are ordered by the `time` column within each sequence. Consecutive actions within a sequence form transition pairs.

For **wide format** data, each row is a sequence and columns represent consecutive time points. Transitions are counted across consecutive columns, skipping any NA values.

Value

A square integer matrix of transition frequencies where $\text{mat}[i, j]$ is the number of times state i was followed by state j . Row and column names are the sorted unique states. Can be passed directly to `tna::tna()`.

See Also

[convert_sequence_format](#) for converting to other representations (frequency counts, one-hot, edge lists).

Examples

```
# Wide format
seqs <- data.frame(V1 = c("A", "B", "A"), V2 = c("B", "A", "C"), V3 = c("A", "C", "B"))
freq <- frequencies(seqs, format = "wide")

# Long format
long <- data.frame(
  Actor = rep(1:2, each = 3), Time = rep(1:3, 2),
  Action = c("A", "B", "C", "B", "A", "C")
)
freq <- frequencies(long, action = "Action", id = "Actor")
```

get_estimator

Retrieve a Registered Estimator

Description

Retrieve a registered network estimator by name.

Usage

```
get_estimator(name)
```

Arguments

name Character. Name of the estimator to retrieve.

Value

A list with elements `fn`, `description`, `directed`.

See Also

[register_estimator](#), [list_estimators](#)

Examples

```
est <- get_estimator("relative")
```

group_regulation_long *Group Regulation in Collaborative Learning (Long Format)*

Description

Students' regulation strategies during collaborative learning, in long format. Contains 27,533 timestamped action records from multiple students working in groups across two courses.

Usage

```
group_regulation_long
```

Format

A data frame with 27,533 rows and 6 columns:

Actor Integer. Student identifier.

Achiever Character. Achievement level: "High" or "Low".

Group Numeric. Collaboration group identifier.

Course Character. Course identifier ("A", "B", or "C").

Time POSIXct. Timestamp of the action.

Action Character. Regulation action (e.g., cohesion, consensus, discuss, synthesis).

Source

Synthetically generated from the group_regulation dataset in the **tna** package.

See Also

[learning_activities](#), [srl_strategies](#)

Examples

```
net <- build_network(group_regulation_long,
  method = "relative",
  actor = "Actor", action = "Action", time = "Time")
net
```

hypergraph_centrality *Hypergraph eigenvector centralities*

Description

Computes one or more eigenvector-style centralities on a [net_hypergraph](#): *clique-motif* (CEC), *Z-eigenvector* (ZEC), and *H-eigenvector* (HEC). Each variant captures influence differently - CEC flattens group structure via clique expansion, while ZEC and HEC propagate through the higher-order groups directly.

Usage

```
hypergraph_centrality(
    hg,
    type = c("clique", "Z", "H"),
    max_iter = 1000L,
    tol = 1e-08,
    normalize = TRUE
)
```

Arguments

hg	A net_hypergraph (from build_hypergraph() or bipartite_groups()).
type	Character vector, any subset of c("clique", "Z", "H"). Default computes all three.
max_iter	Maximum number of power-iteration steps. Default 1000.
tol	Convergence tolerance on the L1 change between successive iterates. Default 1e-8.
normalize	Logical. If TRUE (default), each returned centrality vector is L2-normalized to unit norm (compatible with <code>igraph::eigen centrality()</code> 's scale for type "clique").

Details

Clique-motif eigenvector centrality (CEC): forms the clique-expanded pairwise graph W where $W_{ij} = |\{e : i, j \in e\}|$ and returns the leading eigenvector of W . Equivalent to running `igraph::eigen centrality()` on [clique_expansion\(\)](#) output.

Z-eigenvector centrality (ZEC): solves the linear eigen-equation on the hyperedge tensor,

$$\lambda x_i = \sum_{e \ni i} \prod_{j \in e, j \neq i} x_j,$$

via power iteration. Works for hypergraphs with mixed edge sizes.

H-eigenvector centrality (HEC): solves the power-k-1 eigen-equation,

$$\lambda x_i^{k-1} = \sum_{e \ni i} \prod_{j \in e, j \neq i} x_j.$$

For uniform hypergraphs (all hyperedges of size k), this is equivalent to normalizing the ZEC update by the geometric-mean exponent $1/(k - 1)$. For mixed sizes, the effective exponent is taken from the largest hyperedge; expect slightly different rankings from ZEC in the mixed case.

Value

A named list; one component per requested type. Each component is a named numeric vector of length `hg$n_nodes`.

Note

The "clique" (CEC) variant is validated against `igraph::eigen_centrality` (cosine ~ 1). The "Z" and "H" variants are **(experimental)** - validated only against a clean-room list-based tensor power iteration (same operator, different loop structure); no R package exposes tensor eigenvectors as a primitive for independent comparison.

References

Benson, A. R. (2019). Three hypergraph eigenvector centralities. *SIAM Journal on Mathematics of Data Science* 1(2), 293-312. arXiv:1807.09644.

See Also

[build_hypergraph\(\)](#), [clique_expansion\(\)](#), [hypergraph_measures\(\)](#).

Examples

```
df <- data.frame(
  player = c("A", "B", "C", "A", "B", "D", "C", "D", "E"),
  session = c("S1", "S1", "S1", "S2", "S2", "S3", "S3", "S3", "S3")
)
hg <- bipartite_groups(df, "player", "session")
cent <- hypergraph_centrality(hg)
# Compare rankings across the three variants
do.call(cbind, cent)
```

hypergraph_measures *Structural measures for a hypergraph*

Description

Computes a comprehensive structural-statistics suite for a [net_hypergraph](#): node-level, hyperedge-level, and global measures. All measures are derived in a few BLAS calls on the incidence matrix.

Usage

```
hypergraph_measures(hg)

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

Arguments

hg	A net_hypergraph (from build_hypergraph() or bipartite_groups()).
x	A hypergraph_measures object.
...	Additional arguments (ignored).

Details

All measures are computed via standard matrix operations on the binary incidence $B = (b_{ij})$ where $b_{ij} = 1$ iff node i is in hyperedge j :

- `hyperdegree` = `rowSums(B)`, `edge_sizes` = `colSums(B)`
- `co_degree` = `tcrossprod(B)` (with zero diagonal)
- `edge_pairwise_overlap` = `crossprod(B)` (with zero diagonal)
- `overlap_coefficient[i, j]` = `overlap[i, j] / min(edge_sizes[i], edge_sizes[j])`
- `jaccard[i, j]` = `overlap[i, j] / (edge_sizes[i] + edge_sizes[j] - overlap[i, j])`

Empty hypergraph (`n_hyperedges == 0`) returns trivial zeros and empty matrices.

Value

An object of class `hypergraph_measures` (a named list) with components:

Node-level (length `n_nodes`)

`hyperdegree` Number of hyperedges containing each node.

`node_strength` Total participation: for node i , $\sum_{e \ni i} |e|$. A node in many large hyperedges has high strength.

`max_edge_size` Size of the largest hyperedge containing each node.

`co_degree` `n_nodes` x `n_nodes` matrix: `co_degree[i, j]` = $|\{e : i, j \text{ in } e\}|$ (number of hyperedges co-containing nodes i and j). Diagonal is zero.

Hyperedge-level (length `n_hyperedges` or `m` x `m`)

`edge_sizes` Hyperedge sizes $|e|$.

`edge_pairwise_overlap` `m` x `m` matrix: $|e_i \text{ intersect } e_j|$. Diagonal is zero.

`overlap_coefficient` `m` x `m`: $|e_i \text{ and } e_j| / \min(|e_i|, |e_j|)$. Measures how much the smaller hyperedge is contained in the larger.

`jaccard` `m` x `m`: symmetric overlap index $|e_i \text{ and } e_j| / |e_i \text{ union } e_j|$.

Global (scalars)

`density` For k -uniform hypergraphs: $m / \text{choose}(n, k)$. For mixed sizes: $\text{sum}(|e|) / (n * m)$ (mean fraction of nodes per hyperedge).

avg_edge_size Mean of edge_sizes.
 size_distribution Tabulation of hyperedge sizes (passed through from hg).
 intersection_profile Distribution of pairwise hyperedge intersection sizes - useful for spotting whether hyperedges overlap mostly trivially or share substantial cores (Do et al. 2020).
 pairwise_participation Fraction of node pairs co-appearing in at least one hyperedge.
 n_nodes, n_hyperedges Convenience scalars.

The input x invisibly.

References

Lee, G., Choe, M., & Shin, K. (2024). A survey on hypergraph representation, learning and mining. *Data Mining & Knowledge Discovery* 37, 1-39.
 Do, M. T., Yoon, S., Hooi, B., & Shin, K. (2020). Structural patterns and generative models of real-world hypergraphs. arXiv:2006.07060.

See Also

[build_hypergraph\(\)](#), [bipartite_groups\(\)](#), [clique_expansion\(\)](#).

Examples

```
df <- data.frame(
  player = c("A", "B", "C", "A", "B", "D", "C", "D"),
  session = c("S1", "S1", "S1", "S2", "S2", "S3", "S3", "S3")
)
hg <- bipartite_groups(df, "player", "session")
m <- hypergraph_measures(hg)
print(m)
m$hyperdegree      # how many sessions each player joined
m$co_degree        # pairwise co-membership counts
m$jaccard           # symmetric overlap between sessions
```

learning_activities *Online Learning Activity Indicators*

Description

Simulated binary time-series data for 200 students across 30 time points. At each time point, one or more learning activities may be active (1) or inactive (0). Activities: Reading, Video, Forum, Quiz, Coding, Review. Includes temporal persistence (activities tend to continue across adjacent time points).

Usage

```
learning_activities
```


Format

A data frame with 6,000 rows and 7 columns:

student Integer. Student identifier (1–200).

Reading Integer (0/1). Reading activity indicator.

Video Integer (0/1). Video watching indicator.

Forum Integer (0/1). Discussion forum indicator.

Quiz Integer (0/1). Quiz/assessment indicator.

Coding Integer (0/1). Coding practice indicator.

Review Integer (0/1). Review/revision indicator.

Examples

```
head(learning_activities)
```

list_estimators	<i>List All Registered Estimators</i>
-----------------	---------------------------------------

Description

Return a data frame summarising all registered network estimators.

Usage

```
list_estimators()
```

Value

A data frame with columns name, description, directed.

See Also

[register_estimator](#), [get_estimator](#)

Examples

```
list_estimators()
```

loading_stability

Composite-Weight Stability Under Case Resampling

Description

Experimental. Bootstraps the item weights of a `build_mcml_pc` fit: rows of the raw data are resampled, the node-level network is re-estimated each time, and the connectivity-based composite weights are recomputed. Wide intervals mean the weighting (and therefore the "loadings" macro network) should not be over-interpreted.

Usage

```
loading_stability(x, iter = 200L, ci_level = 0.05, seed = NULL)
```

Arguments

<code>x</code>	An <code>mcml_pc</code> object that carries raw data.
<code>iter</code>	Integer. Bootstrap replicates (default 200; node-level re-estimation makes this heavier than a plain bootstrap).
<code>ci_level</code>	Numeric. Significance level for percentile CIs (default 0.05).
<code>seed</code>	Integer or NULL. RNG seed.

Value

An object of class "pc_loading_stability": a list with `summary` (tidy data frame: node, cluster, weight, boot_mean, boot_sd, ci_lower, ci_upper, sign_flips - the proportion of replicates in which the item's sign differed from the observed one), `boot_weights` (iter x n_nodes matrix), `iter`, and `ci_level`. Has `print` and `plot` methods.

Examples

```
set.seed(1)
df <- as.data.frame(matrix(rnorm(600), 100, 6))
names(df) <- c("a1", "a2", "a3", "b1", "b2", "b3")
cl <- list(A = c("a1", "a2", "a3"), B = c("b1", "b2", "b3"))
fit <- build_mcml_pc(df, cl, aggregation = "loadings",
  method = "cor")
ls <- loading_stability(fit, iter = 50, seed = 1)
ls$summary
```

long-data	<i>Human-AI Vibe Coding Interaction Data (Long Format)</i>
-----------	--

Description

Coded interaction sequences from 429 human-AI pair programming sessions across 34 projects, in long format.

Usage

human_long

ai_long

Format

Data frames in long format with 9 columns:

message_id Integer. Turn index.

project Character. Project identifier (Project_1 .. Project_34).

session_id Character. Unique session hash.

timestamp Integer. Unix timestamp for ordering.

session_date Character. Date of the session (YYYY-MM-DD).

code Character. Interaction code (17 action labels).

cluster Character. High-level cluster: Action, Communication, Directive, Evaluative, Metacognitive, or Repair.

code_order Integer. Order of the code within the session.

order_in_session Integer. Absolute turn order within the session.

human_long	Human turns only, 10,796 rows
ai_long	AI turns only, 8,551 rows

An object of class `data.frame` with 10796 rows and 9 columns.

An object of class `data.frame` with 8551 rows and 9 columns.

Source

Saqr, M. (2026). Human-AI vibe coding interaction study. <https://saqr.me/blog/2026/human-ai-interaction-cograp>

Examples

```
net <- build_network(human_long, method = "tna",
                    action = "code", actor = "session_id",
                    time = "timestamp")

net
```

`long_to_wide`*Convert Long Format to Wide Sequences*

Description

Convert sequence data from long format (one row per action) to wide format (one row per sequence, columns as time points).

Usage

```
long_to_wide(  
  data,  
  id_col = "Actor",  
  time_col = "Time",  
  action_col = "Action",  
  time_prefix = "V",  
  fill_na = TRUE  
)
```

Arguments

<code>data</code>	Data frame in long format.
<code>id_col</code>	Character. Name of the column identifying sequences. Default: "Actor".
<code>time_col</code>	Character. Name of the column identifying time points. Default: "Time".
<code>action_col</code>	Character. Name of the column containing actions/states. Default: "Action".
<code>time_prefix</code>	Character. Prefix for time point columns in output. Default: "V".
<code>fill_na</code>	Logical. Whether to fill missing time points with NA. Default: TRUE.

Details

This function converts long format data (like that from `simulate_long_data()`) to the wide format expected by `tna::tna()` and related functions.

If `time_col` contains non-integer values (e.g., timestamps), the function will use the ordering within each sequence to create time indices.

Value

A data frame in wide format where each row is a sequence and columns `V1`, `V2`, ... contain the actions at each time point.

See Also

[wide_to_long](#) for the reverse conversion, [prepare_for_tna](#) for preparing data for TNA analysis.

Examples

```
long_data <- data.frame(
  Actor = rep(1:3, each = 4),
  Time = rep(1:4, 3),
  Action = sample(c("A", "B", "C"), 12, replace = TRUE)
)
wide_data <- long_to_wide(long_data, id_col = "Actor")
head(wide_data)
```

magnitude_difference	<i>Magnitude difference between the frequency and probability views</i>
----------------------	---

Description

Quantifies, per edge, how much row-normalization moves a transition network between its two natural summaries: raw transition counts (frequency / FTNA, `build_network(method = "frequency")`) and row-conditional probabilities (TNA, `build_network(method = "relative")`). The two matrices rank edges differently - an edge that is large in counts can be modest in probability, and a rare-source edge can dominate its row in probability. The per-edge discrepancy on a common scale is the *magnitude difference*.

Usage

```
magnitude_difference(
  data,
  actor = "Actor",
  action = "Action",
  time = NULL,
  metric = c("abs_diff", "chord_dist", "atanh_diff", "geom_norm_diff", "cv_inflation"),
  scale = c("tna_range", "rank_minmax", "minmax", "none"),
  format = c("auto", "long", "wide")
)

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

## S3 method for class 'magnitude_difference'
plot(x, type = c("stacked", "circular"), min_show = 0.01, title = NULL, ...)
```

Arguments

`data` Long- or wide-format event log (`data.frame`).

`actor, action, time` Column names in data (long format). `time` may be `NULL`.

metric	Discrepancy metric. One of "abs_diff" (default; absolute difference, simplest and most robust), "chord_dist" (chord distance on the unit sphere), "atanh_diff" (Fisher z-style on a bounded scale), "geom_norm_diff" (geometric-mean normalized; amplifies small edges), "cv_inflation" (per-vector SD-standardized then absolute difference).
scale	How the two weight matrices are placed on a common scale before differencing. "tna_range" (default) rescales FTNA linearly into TNA's [min, max] range and leaves TNA untouched, so the difference is in TNA probability units. "rank_minmax" converts each matrix's values to ranks scaled to [0, 1] (ordinal). "minmax" scales each matrix's raw values to [0, 1] separately (asymmetric - TNA's max and FTNA's max map to the same value despite differing native ranges). "none" uses raw weights.
format	Input format passed through to <code>build_network()</code> ; "auto" (default) treats the data as wide when action is not a column.
x	A magnitude_difference object.
...	Passed to plotting helpers (ignored by print).
type	Plot style, "stacked" (default) or "circular".
min_show	For type = "circular", drop edges whose magnitude is below this fraction of the maximum.
title	Plot title. NULL generates one from the metric and scale.

Value

An object of class "magnitude_difference": a list with \$edges (per-edge data.frame with columns from, to, ftna, tna, signed = tna - ftna, and value = the chosen metric), \$metric, \$scale, \$weights_ftna, \$weights_tna, and \$states.

print invisibly returns x.

plot returns a ggplot object.

Methods (by generic)

- `print(magnitude_difference)`: Print a compact summary of the per-edge magnitude-difference distribution.
- `plot(magnitude_difference)`: Plot the per-edge magnitude difference as a polar portrait. `type = "stacked"` (default) draws one sector per from-state with stacked wedges (grey base = shared value, colored tip = magnitude difference); `type = "circular"` draws a chord-style diagram with signed differences on a diverging blue-orange scale.

See Also

`build_network()`, `compare_model()`

Examples

```
data(group_regulation_long, package = "Nestimate")
fit <- magnitude_difference(group_regulation_long,
```

```

        actor = "Actor", action = "Action",
        time = "Time")

print(fit)
# Edges most promoted by row-normalization (rare-source transitions):
head(fit$edges[order(-fit$edges$signed), c("from", "to", "ftna", "tna")])

plot(fit)                # stacked polar portrait
plot(fit, type = "circular") # chord-style diagram

```

markov_order_test	<i>Test the Markov order of a sequential process</i>
-------------------	--

Description

Principled test of whether a categorical sequence is best described as a k -th order Markov chain. At each order $k = 1, \dots, \text{max_order}$, the function computes the classical likelihood-ratio statistic (G^2) for the conditional independence $s \perp x \mid w$, where w is the $(k - 1)$ -gram context, x is the extra (k -th-back) state added at order k , and s is the next state. Under H_0 (order- $(k - 1)$ is correct), s is independent of x given w .

The null distribution is obtained by an **exact within- w permutation test**: for each context w the successor labels are exchangeable under H_0 , so shuffling s within each w -group yields an exact reference distribution for G^2 . No plug-in MLE bias and no refitting per replicate. An asymptotic χ^2 p-value is reported alongside for reference.

The optimal order is the smallest k that is **not** significantly better than $k - 1$ at level alpha: we keep raising the order while the test rejects, and stop at the first non-rejection.

Usage

```

markov_order_test(
  data,
  max_order = 3L,
  n_perm = 500L,
  alpha = 0.05,
  parallel = FALSE,
  n_cores = 2L,
  seed = NULL
)

```

Arguments

data	A data.frame (wide format, one sequence per row) or list of character vectors (one per trajectory). NAs are treated as end of sequence.
max_order	Integer. Highest Markov order to test. Default 3.
n_perm	Integer. Number of within- w permutations per order. Default 500.
alpha	Numeric. Significance level for order selection. Default 0.05.

<code>parallel</code>	Logical. Use <code>parallel::mclapply</code> for permutations. Default FALSE (set TRUE only on Unix-like systems).
<code>n_cores</code>	Integer. Cores for parallel execution. Default 2.
<code>seed</code>	Optional integer seed for reproducibility.

Value

An object of class `net_markov_order` with elements:

optimal_order Integer. Selected order via sequential permutation test.

bic_order Integer. Order minimising BIC (reported by print/plot; not used in the permutation selection).

aic_order Integer. Order minimising AIC (reported by print/plot; not used in the permutation selection).

test_table Tidy data.frame, one row per order tested with columns `order`, `loglik`, `AIC`, `BIC`, `df`, `g2`, `p_permutation`, `p_asymptotic`, `significant`. AIC/BIC are the information-criterion values used for the `ic` plot panel and to derive `aic_order`/`bic_order`; the order-0 row has NA for the test columns (`df`, `g2`, `p_permutation`, `p_asymptotic`, `significant`).

permutation_null List of numeric vectors (length `max_order`), one empirical null G^2 distribution per order.

logliks Named numeric vector of log-likelihoods per order (for AIC / BIC panel only, not used in the test).

layer_dofs Named integer vector of model degrees of freedom per order (free parameters added at each layer), used to compute the AIC / BIC columns.

transition_matrices List of fitted transition matrices.

states Character vector of observed state labels.

n_sequences, n_observations Data summary.

n_perm, alpha, max_order Call settings.

Examples

```
# First-order Markov data: test should select order 1
set.seed(1)
states <- letters[1:4]
tm <- matrix(runif(16), 4, 4, dimnames = list(states, states))
tm <- tm / rowSums(tm)
seqs <- lapply(1:30, function(.) {
  s <- character(50); s[1] <- sample(states, 1)
  for (i in 2:50) s[i] <- sample(states, 1, prob = tm[s[i - 1], ])
  s
})
res <- markov_order_test(seqs, max_order = 3, n_perm = 300, seed = 1)
res$optimal_order
summary(res)
plot(res)
```


Description

Computes per-state stability metrics from a transition network: persistence (self-loop probability), stationary distribution, mean recurrence time, sojourn time, and mean accessibility to/from other states.

Usage

```
markov_stability(x, normalize = TRUE)

## S3 method for class 'net_markov_stability'
plot(
  x,
  metrics = c("persistence", "stationary_prob", "return_time", "sojourn_time",
    "avg_time_to_others", "avg_time_from_others"),
  combined = TRUE,
  ...
)
```

Arguments

x	A netobject, cograph_network, tna object, row-stochastic numeric transition matrix, or a wide sequence data.frame (rows = actors, columns = time-steps).
normalize	Logical. Normalize rows to sum to 1? Default TRUE.
metrics	Character vector. Which metrics to plot. Options: "persistence", "stationary_prob", "return_time", "sojourn_time", "avg_time_to_others", "avg_time_from_others". Default: all six.
combined	When TRUE (default), all selected metrics are shown in one ggplot via facet_wrap(~metric). When FALSE, returns a named list of single-panel ggplots, one per metric, so each can be printed, saved, or re-laid-out independently.
...	Ignored.

Details

Sojourn time is the expected consecutive time steps spent in a state before leaving: $1/(1 - P_{ii})$. States with persistence = 1 have sojourn_time = Inf.

avg_time_to_others: mean passage time from this state to all others; reflects how "sticky" or "isolated" the state is.

avg_time_from_others: mean passage time from all other states to this one; reflects accessibility (attractor strength).

Value

An object of class "net_markov_stability" with:

stability Data frame with one row per state and columns: state, persistence (P_{ii}), stationary_prob (π_i), return_time ($1/\pi_i$), sojourn_time ($1/(1 - P_{ii})$), avg_time_to_others (mean MFPT leaving state i), avg_time_from_others (mean MFPT arriving at state i).

mpt The underlying net_mpt object.

See Also

[passage_time](#)

Examples

```
net <- build_network(as.data.frame(trjectories), method = "relative")
ms <- markov_stability(net)
print(ms)

plot(ms)
```

mark_first_state	<i>Mark leading-NA cells with an explicit state label</i>
------------------	---

Description

Mirror of [mark_terminal_state\(\)](#) for *left-censored* sequence data. Replaces every cell *before* each row's first observed state with the label given by state. The resulting chain has a structurally *recurrent* "Start" state that everyone enters from - useful for cohort-entry analyses where students join at different time points and you want a uniform pre-observation marker.

Usage

```
mark_first_state(data, state = "Start", cols = NULL)
```

Arguments

data	A wide-format matrix or data.frame (rows = actors, cols = time steps) of state labels with NA for missing observations.
state	Character. Label to insert in leading-NA cells. Default "Start".
cols	Optional state-column names; otherwise all columns.

Details

Unlike [mark_terminal_state\(\)](#), the marked state is **not** absorbing in the resulting transition matrix - every transition from "Start" goes to one of the original states (the actor's first observed state), and the "Start" row is row-stochastic exactly as the data dictates.

Value

A data.frame of the same shape as data with leading NAs filled by state.

See Also

[mark_terminal_state\(\)](#), [actor_endpoints\(\)](#)

Examples

```
M <- mark_first_state(trjectories, state = "Start")
# In a chain built from M, "Start" is a transient entry point.
```

mark_terminal_state	<i>Mark terminal-NA cells with an explicit state label</i>
---------------------	--

Description

Replaces every cell after each row's last observed state with the label given by state, leaving non-terminal NAs untouched. The result, passed to [build_network\(\)](#), yields a Markov chain in which the marked state is **absorbing** by construction ($P[\text{state}, \text{state}] = 1$).

Usage

```
mark_terminal_state(data, state = "End", cols = NULL)
```

Arguments

data	A wide-format matrix or data.frame (rows = actors, cols = time steps) of state labels with NA for missing observations.
state	Character. Label to insert in terminal-NA cells. Default "End".
cols	Optional state-column names; otherwise all columns.

Details

This is the small piece of pre-processing required to turn right-censored sequence data into an absorbing-chain model. The chain on the resulting matrix has one extra state (state) which is structurally absorbing because every cell after the actor's last observed step has been set to state - the chain stays there forever once entered.

Use [chain_structure\(\)](#) on the result to compute mean absorption time, absorption probabilities, and per-state classification. Note that [markov_stability\(\)](#) is *not* the right summary for absorbing chains; its stationary distribution will collapse to the absorbing state.

Value

A data.frame of the same shape as data with terminal NAs filled by state.

See Also

`actor_endpoints()`, `chain_structure()`, `build_network()`

Examples

```
M <- mark_terminal_state(trajectories, state = "Dropout")
net <- build_network(M, method = "relative")
chain_structure(net)
```

mogen_transitions

Extract Transition Table from a MOGen Model

Description

Returns a data frame of all transitions at a given Markov order, sorted by count (descending). Each row shows the full path as a readable sequence of states, along with the observed count and transition probability.

Usage

```
mogen_transitions(x, order = NULL, min_count = 1L)
```

Arguments

<code>x</code>	A <code>net_mogen</code> object from <code>build_mogen()</code> .
<code>order</code>	Integer. Which order's transitions to extract. Must be a whole number; a non-integer value is an error rather than being silently truncated. Defaults to the optimal order selected by the model.
<code>min_count</code>	Integer. Minimum observed count to include (default 1). Use this to filter out rare transitions that have unreliable probabilities.

Details

At order k , each edge in the De Bruijn graph represents a $(k+1)$ -step path. For example, at order 2, the edge from node "AI -> FAIL" to node "FAIL -> SOLVE" represents the three-step path AI -> FAIL -> SOLVE. The path column reconstructs this full sequence for readability.

Value

A data frame with columns:

path The full state sequence (e.g., "AI -> FAIL -> SOLVE").

count Number of times this transition was observed.

probability Transition probability $P(\text{to} \mid \text{from})$.

from The context / conditioning states (k -gram source node).

to The predicted next state.

Examples

```
seqs <- list(c("A","B","C","D"), c("A","B","C","A"), c("B","C","D","A"))
mg <- build_mogen(seqs, max_order = 2)
mogen_transitions(mg, order = 1)

trajs <- list(c("A","B","C","D"), c("A","B","D","C"),
             c("B","C","D","A"), c("C","D","A","B"))
m <- build_mogen(trajs, max_order = 3)
mogen_transitions(m, order = 1)
```

mosaic_analysis	<i>Two-variable mosaic analysis (chi-square test + flat mosaic)</i>
-----------------	---

Description

Analyses the association between two categorical columns of a `data.frame`. Builds the contingency table, drops sparse categories below `min_count`, runs a Pearson chi-square test (or Fisher's exact test), computes Cramer's V with a df-adjusted effect-size label, and draws a flat `ggplot2` mosaic whose tile area encodes counts and whose fill encodes the standardized Pearson residual (Nestimate diverging palette). All tabular output is a tidy one-row-per-cell `data.frame`.

Usage

```
mosaic_analysis(
  data,
  var1,
  var2,
  min_count = 10L,
  test = c("chisq", "fisher"),
  percentage_base = c("total", "row", "column"),
  tile_label = c("count", "percent", "residual", "category", "none"),
  title = "",
  ...
)
```

Arguments

<code>data</code>	A <code>data.frame</code> containing the two variables.
<code>var1</code>	Character. Name of the first variable (mosaic columns).
<code>var2</code>	Character. Name of the second variable (stacked within columns).
<code>min_count</code>	Integer. Minimum marginal count for a category to be kept. Categories of either variable below this are dropped before testing. Default 10.

<code>test</code>	Character. "chisq" (default) Pearson chi-square, or "fisher" Fisher's exact test (simulated p-value). Cramer's V and the residual fill are always derived from the chi-square statistic.
<code>percentage_base</code>	Character. Base for the "percent" tile label and the pct column: "total" (default), "row" (within var1), or "column" (within var2).
<code>tile_label</code>	Character. What to print inside each tile: "count" (default), "percent", "residual", "category" (var2 level), or "none".
<code>title</code>	Character. Plot title. Default "".
<code>...</code>	Further flat-mosaic styling arguments passed to the renderer (e.g. <code>col_label_side</code> , <code>row_label_side</code> , <code>legend_position</code> , <code>legend_size</code> , <code>label_size</code> , <code>palette</code>). Tile fill uses the ColorBrewer RdBu ramp by default (override with <code>palette</code>). Column labels auto-rotate to vertical when there are more than 6 columns; pass <code>col_label_angle</code> to force an angle.

Value

An object of class "mosaic_analysis": a list with

plot The flat mosaic ggplot object.

counts Tidy data.frame, one row per (var1, var2) cell, with observed, expected, residual (standardized), and pct (on percentage_base).

stats One-row data.frame: test, statistic, df, p_value, cramers_v, effect_size, n.

test The raw htest object.

cramers_v, effect_size Effect size value and label.

table The filtered contingency table.

removed List of dropped var1 / var2 categories.

n_original, n_filtered Row counts before/after filtering.

See Also

[mosaic_plot](#) for the network/table mosaic (which also accepts `style = "flat"`).

Examples

```
df <- data.frame(
  gender = sample(c("F", "M"), 200, replace = TRUE),
  level = sample(c("Low", "Mid", "High"), 200, replace = TRUE)
)
res <- mosaic_analysis(df, "gender", "level", min_count = 5)
res$stats
res$counts

plot(res, tile_label = "percent")
```

mosaic_plot

*Mosaic Plot of a Network's Transition or Co-occurrence Counts***Description**

Draws a Hartigan-Friendly mosaic (marimekko geometry, chi-square standardized-residual fill) for an integer-weighted network. Equivalent in algorithm and appearance to `tna::plot_mosaic()`; named differently to avoid an export clash when both packages are attached.

Usage

```
mosaic_plot(x, ...)

## Default S3 method:
mosaic_plot(x, ...)

## S3 method for class 'netobject'
mosaic_plot(
  x,
  xlab = NULL,
  ylab = NULL,
  range = NULL,
  top_angle = NULL,
  left_angle = NULL,
  residuals = c("permutation", "asymptotic"),
  n_perm = 500L,
  seed = NULL,
  values = FALSE,
  style = c("classic", "flat"),
  ...
)

## S3 method for class 'htna'
mosaic_plot(
  x,
  xlab = NULL,
  ylab = NULL,
  range = NULL,
  top_angle = NULL,
  left_angle = NULL,
  residuals = c("permutation", "asymptotic"),
  n_perm = 500L,
  seed = NULL,
  values = FALSE,
  style = c("classic", "flat"),
  ...
)
```

```
## S3 method for class 'mcmc1'
mosaic_plot(
  x,
  level = c("macro", "clusters"),
  xlab = NULL,
  ylab = NULL,
  range = NULL,
  top_angle = NULL,
  left_angle = NULL,
  residuals = c("permutation", "asymptotic"),
  n_perm = 500L,
  seed = NULL,
  ncol = 2L,
  values = FALSE,
  style = c("classic", "flat"),
  ...
)

## S3 method for class 'netobject_group'
mosaic_plot(
  x,
  xlab = NULL,
  ylab = NULL,
  range = NULL,
  top_angle = NULL,
  left_angle = NULL,
  residuals = c("permutation", "asymptotic"),
  n_perm = 500L,
  seed = NULL,
  ncol = 2L,
  values = FALSE,
  style = c("classic", "flat"),
  ...
)

## S3 method for class 'table'
mosaic_plot(
  x,
  xlab = "Row",
  ylab = "Column",
  range = NULL,
  top_angle = NULL,
  left_angle = NULL,
  residuals = c("permutation", "asymptotic"),
  n_perm = 500L,
  seed = NULL,
  values = FALSE,
```



```

    style = c("classic", "flat"),
    ...
)

## S3 method for class 'matrix'
mosaic_plot(x, ...)

```

Arguments

<code>x</code>	One of the four data-bearing Nestimate classes: <code>netobject</code> (single mosaic of <code>\$weights</code>), <code>netobject_group</code> (one panel per group), <code>mcml</code> (between-cluster mosaic by default; per-cluster panels with <code>level = "within"</code>), or <code>htna</code> (single mosaic of <code>\$weights</code> ; <code>htna</code> inherits <code>netobject</code> so the geometry matches). Also accepts a contingency table or plain numeric matrix for ad-hoc plotting.
<code>...</code>	Flat styling overrides forwarded to the flat renderer when <code>style = "flat"</code> (otherwise ignored).
<code>xlab, ylab</code>	Axis labels. <code>NULL</code> (default) draws no axis title. Pass any string to add one.
<code>range</code>	Numeric of length 2 giving the lower and upper colour-scale limits for the standardized residual. <code>NULL</code> (default) auto-fits the limits to the symmetric range <code>c(-M, M)</code> where <code>M = max(stdres)</code> , so no signal is squished. Pass an explicit range (e.g. <code>c(-4, 4)</code> for <code>tna</code> -style display, <code>c(-6, 6)</code> for moderate clipping) to clamp the colour scale.
<code>top_angle, left_angle</code>	Rotation in degrees for the top (x) and left (y) tick labels. <code>NULL</code> (default) uses the auto rule 90 if <code>n_levels > 3</code> else 0 on each axis. Pass any numeric to override (e.g. <code>top_angle = 45</code> , <code>left_angle = 0</code>).
<code>residuals</code>	One of "permutation" (default) or "asymptotic". "permutation" computes empirical-null z-scores by shuffling one variable's labels against the other for <code>n_perm</code> draws and reporting $(0 - \text{mean_perm}) / \text{sd_perm}$ per cell. Robust on sparse tables. "asymptotic" returns <code>stats::chisq.test()\$stdres</code> (the closed-form $(0 - E) / \sqrt{E * (1 - p_row) * (1 - p_col)}$) that <code>vcd</code> and <code>tna</code> use).
<code>n_perm</code>	Number of permutations when <code>residuals = "permutation"</code> . Default 500; use ≥ 1000 for stable tail estimates.
<code>seed</code>	Optional integer seed for the permutation RNG. Use for reproducible plots; ignored when <code>residuals = "asymptotic"</code> .
<code>values</code>	Logical. When <code>TRUE</code> , overlay each cell's standardized residual as a numeric label (one decimal). Text colour switches to white on saturated cells (<code>lstdresl > 1.5</code>) and dark grey otherwise. Default <code>FALSE</code> – the colour bar legend already conveys the sign and magnitude.
<code>style</code>	Character. "classic" (default) draws the black-bordered <code>theme_minimal</code> mosaic with axis labels; "flat" draws the flat <code>theme_void</code> mosaic with white gutters, a soft background and a colour-bar legend. With <code>style = "flat"</code> , <code>values = TRUE</code> prints the standardized residual inside each tile, and extra flat styling arguments (<code>tile_label</code> , <code>col_label_side</code> , <code>row_label_side</code> , <code>legend_position</code> , <code>legend_size</code> , <code>label_size</code> , ...; see mosaic_analysis) may be passed via Not supported for multi-panel <code>mcml</code> (<code>level = "clusters"</code>), which falls back to the classic faceted style.

level	For mcml only. "macro" (default) draws a single mosaic of x\$macro\$weights (the cluster-by-cluster aggregate); "clusters" draws one mosaic per cluster from x\$clusters[[k]]\$weights, faceted into one combined ggplot.
ncol	For netobject_group: number of columns in the small- multiples layout. Default 2.

Details

Column widths are proportional to row marginals of the weight matrix (incoming totals when the matrix is transposed, as for transitions). Within each column, segment heights are proportional to that row’s conditional distribution. Cell fill is the standardized residual from `stats::chisq.test()`, with a diverging palette clipped to ± 4 . Mosaics need integer counts: when `$weights` is already integer (method = "frequency" / "co_occurrence") it is used directly; for a single netobject / ht na otherwise (relative, glasso, cor, ...) order-1 transition counts are recounted from the raw `$data` sequences. The function errors only when neither integer weights nor `$data` are available.

Value

A ggplot object (or a gtable from `gridExtra::arrangeGrob` for netobject_group when **gridExtra** is available).

See Also

[plot_mosaic](#) for the lower-level data.frame primitive.

Examples

```
## Not run:
net <- build_network(group_regulation, method = "frequency")
mosaic_plot(net)

## End(Not run)
```

nct	<i>Network Comparison Test</i>
-----	--------------------------------

Description

Tests whether two networks estimated from independent samples differ at three levels: **global strength** (M-statistic), **network structure** (S-statistic, max absolute edge difference), and **individual edges** (E-statistic per edge). Inference is via permutation of group labels.

Usage

```
nct(
  data1,
  data2,
  iter = 1000L,
  gamma = 0.5,
  paired = FALSE,
  abs = TRUE,
  weighted = TRUE,
  p_adjust = "none"
)
```

Arguments

<code>data1</code>	A numeric matrix or data.frame of observations from group 1.
<code>data2</code>	A numeric matrix or data.frame of observations from group 2. Same number of columns as <code>data1</code> .
<code>iter</code>	Integer. Number of permutation iterations. Default 1000.
<code>gamma</code>	EBIC tuning parameter for glasso. Default 0.5.
<code>paired</code>	Logical. If TRUE, perform a paired permutation (within-subject swap). Default FALSE.
<code>abs</code>	Logical. If TRUE, compute global strength on absolute edge weights. Default TRUE.
<code>weighted</code>	Logical. If TRUE, use weighted networks for the tests. If FALSE, binarize before computing statistics. Default TRUE.
<code>p_adjust</code>	P-value adjustment method for the per-edge tests (any method in <code>stats::p.adjust.methods</code>). Default "none".

Details

Implementation matches `NetworkComparisonTest::NCT()` with defaults `abs = TRUE`, `weighted = TRUE`, `paired = FALSE` at machine precision when the same seed is used. The network estimator is EBIC-selected glasso applied to a Pearson correlation matrix, with `Matrix::nearPD` symmetrization (matching NCT's `NCT_estimator_GGM` default).

Value

A list of class `net_nct` with elements:

- nw1, nw2** Estimated weighted adjacency matrices.
- M** List with observed, perm, `p_value` for the global strength test.
- S** Same structure for the maximum absolute edge difference.
- E** Same structure for per-edge tests.
- n_iter** Number of permutations.
- paired** Whether a paired test was used.

Examples

```
## Not run:
set.seed(1)
x1 <- matrix(rnorm(200 * 5), 200, 5)
x2 <- matrix(rnorm(200 * 5), 200, 5)
colnames(x1) <- colnames(x2) <- paste0("V", 1:5)
res <- nct(x1, x2, iter = 100)
res$M$p_value
res$S$p_value

## End(Not run)
```

network_reliability *Split-Half Reliability for Network Estimates*

Description

Assesses the stability of network estimates by repeatedly splitting sequences into two halves, building networks from each half, and comparing them. Supports single-model reliability assessment and multi-model comparison with optional scaling for cross-method comparability.

For transition methods ("relative", "frequency", "co_occurrence"), uses pre-computed per-sequence count matrices for fast resampling (same infrastructure as [bootstrap_network](#)).

Usage

```
network_reliability(
  ...,
  iter = 1000L,
  split = 0.5,
  scale = "none",
  seed = NULL
)
```

Arguments

...	One or more netobjects (from build_network). If unnamed, each model is auto-named from its \$method. A netobject_group is flattened into its constituent models.
iter	Integer. Number of split-half iterations (default: 1000).
split	Numeric. Fraction of sequences assigned to the first half (default: 0.5).
scale	Character. Scaling applied to both split-half matrices before computing metrics. One of "none" (default), "minmax", "standardize", or "proportion". Use scaling when comparing models on different scales (e.g. frequency vs relative).
seed	Integer or NULL. RNG seed for reproducibility.

Value

An object of class "net_reliability" containing:

iterations Data frame with columns model, mean_dev, median_dev, cor, max_dev (one row per iteration per model).

summary Data frame with columns model, metric, mean, sd.

models Named list of the original netobjects.

iter Number of iterations.

split Split fraction.

scale Scaling method used.

See Also

[build_network](#), [bootstrap_network](#)

Examples

```
net <- build_network(data.frame(V1 = c("A","B","C","A"),
  V2 = c("B","C","A","B")), method = "relative")
rel <- network_reliability(net, iter = 10)

seqs <- data.frame(
  V1 = sample(LETTERS[1:4], 30, TRUE), V2 = sample(LETTERS[1:4], 30, TRUE),
  V3 = sample(LETTERS[1:4], 30, TRUE), V4 = sample(LETTERS[1:4], 30, TRUE)
)
net <- build_network(seqs, method = "relative")
rel <- network_reliability(net, iter = 100, seed = 42)
print(rel)
```

net_aggregate_weights *Aggregate Edge Weights*

Description

Aggregates a vector of edge weights using various methods. Compatible with igraph's edge.attr.comb parameter.

Usage

```
net_aggregate_weights(w, method = "sum", n_possible = NULL)
```

Arguments

w	Numeric vector of finite edge weights. NA and zero weights are excluded before aggregation, so every method (including "density", "min", "max", "prod", "geomean") operates on the non-zero, non-NA subset.
method	Single aggregation method: "sum", "mean", "median", "max", "min", "prod", "density", or "geomean". Because zeros are stripped first, "density" ($\text{sum}(w) / n_{\text{possible}}$) and "mean" ($\text{sum}(w) / \text{number of non-zero edges}$) return the <i>same</i> value whenever the block is fully dense – i.e. when the count of non-zero edges equals n_{possible} . They diverge only when zero/NA edges are present (then "density" divides by the larger n_{possible} , "mean" by the smaller non-zero count).
n_possible	Optional single finite numeric number of possible edges for density calculation. When omitted, "density" falls back to $\text{sum}(w) / \text{length}(w)$ on the non-zero subset (equivalent to "mean"); supply n_{possible} (e.g. the block size $n_i * n_j$) for a true edge density.

Value

Single aggregated value

Examples

```
w <- c(0.5, 0.8, 0.3, 0.9)
net_aggregate_weights(w, "sum")    # 2.5
net_aggregate_weights(w, "mean")  # 0.625
net_aggregate_weights(w, "max")   # 0.9
net_aggregate_weights(w, "density", n_possible = 9) # 2.5 / 9
```

net_centrality

Compute Centrality Measures for a Network

Description

Computes centrality measures from a `netobject`, `netobject_group`, `mcml`, or `cograph_network`. The built-in measures match `tna::centralities()` without importing `tna` or `igraph`. The only intentional default difference is that Diffusion is range-normalized by default.

Usage

```
net_centrality(
  x,
  measures = NULL,
  loops = FALSE,
  normalize = FALSE,
  invert = TRUE,
  normalize_diffusion = TRUE,
  centrality_fn = NULL,
```

```
    ...  
  )
```

Arguments

x	A netobject, netobject_group, or cograph_network.
measures	Character vector. Centrality measures to compute. Defaults to c("InStrength", "Betweenness", "Diffusion"). Pass "all" for every built-in measure: "OutStrength", "InStrength", "ClosenessIn", "ClosenessOut", "Closeness", "Betweenness", "BetweennessRSP", "Diffusion", and "Clustering". The legacy aliases "InCloseness" and "OutCloseness" are also accepted.
loops	Logical. Include self-loops (diagonal) in computation? Default: FALSE.
normalize	Logical. Range-normalize all requested measures using the same transformation as tna::centralities(normalize = TRUE). Default: FALSE.
invert	Logical. Invert weights for shortest-path measures? Default: TRUE, matching tna.
normalize_diffusion	Logical. Range-normalize Diffusion even when normalize = FALSE. Default: TRUE.
centrality_fn	Optional function. Custom centrality function that takes a weight matrix and returns a named list of centrality vectors.
...	Additional arguments (ignored).

Value

For a netobject: a net centrality data frame with node names as rows, a state column, and one column per centrality measure. For a netobject_group: a net centrality_group list of such data frames.

Examples

```
seqs <- data.frame(  
  V1 = c("A", "B", "A", "C"), V2 = c("B", "C", "B", "A"),  
  V3 = c("C", "A", "C", "B"))  
net <- build_network(seqs, method = "relative")  
net Centrality(net)
```

net_deprune

Undo Network Pruning

Description

Restores the original (pre-pruning) weights of a network pruned by [net_prune](#), without recomputation. The pruning record is kept, so [net_reprune](#) can re-apply it.

Usage

```
net_deprune(x, ...)

## S3 method for class 'netobject'
net_deprune(x, ...)

## S3 method for class 'netobject_group'
net_deprune(x, ...)

## Default S3 method:
net_deprune(x, ...)
```

Arguments

x	A pruned netobject or netobject_group.
...	Ignored.

Value

The network (or group) with original weights restored and its pruning marked inactive.

See Also

[net_prune](#), [net_reprune](#)

Examples

```
seqs <- data.frame(V1 = c("A", "B", "C"), V2 = c("B", "C", "A"),
                   V3 = c("C", "A", "B"))
net    <- build_network(seqs, method = "relative")
pruned <- net_prune(net, threshold = 0.2)
net_deprune(pruned)
```

net_edge_betweenness *Edge Betweenness Network*

Description

Builds a network in which each edge's weight is replaced by its betweenness: the number of shortest paths between all node pairs that traverse that edge (fractional when shortest paths tie). This is the Nestimate counterpart of `tna::betweenness_network()` and produces identical values for transition networks; the name differs to avoid a clash with `tna::betweenness_network()` and `igraph::edge_betweenness()`.

Usage

```
net_edge_betweenness(x, invert = TRUE, ...)

## S3 method for class 'netobject'
net_edge_betweenness(x, invert = TRUE, ...)

## S3 method for class 'netobject_group'
net_edge_betweenness(x, invert = TRUE, ...)

## Default S3 method:
net_edge_betweenness(x, invert = TRUE, ...)
```

Arguments

<code>x</code>	A <code>netobject</code> or <code>netobject_group</code> .
<code>invert</code>	Logical. Invert weights to distances by $1/w$ before computing shortest paths? Default TRUE (correct for probability and frequency networks).
<code>...</code>	Additional arguments (ignored).

Details

For a probability/transition network the edge weights are transition probabilities, so they are inverted to distances (`invert = TRUE`) before path-finding: the geodesic between two states is then the most probable route rather than the one with the fewest hops. Pass `invert = FALSE` when the weights already represent distances.

Directedness is taken from the network itself. A directed network yields an asymmetric betweenness matrix; an undirected (symmetric) network yields a symmetric one.

Value

For a `netobject`: a new `netobject` (class `c("netobject", "cograph_network")`) whose `$weights` are the edge-betweenness scores, with `method = "edge_betweenness"`. Call `extract_edges()` on it for a tidy per-edge table, or `plot()` to render it. The object preserves source-network metadata so [permutation](#) can test edge-betweenness differences by permuting the source networks. For a `netobject_group`: a `netobject_group` of such networks, one per group.

Examples

```
seqs <- data.frame(
  V1 = c("A", "B", "A", "C"), V2 = c("B", "C", "B", "A"),
  V3 = c("C", "A", "C", "B"))
net <- build_network(seqs, method = "relative")
eb <- net_edge_betweenness(net)
extract_edges(eb)
```

net_prune

Prune a Network's Edges

Description

Removes weak or non-significant edges from a network, keeping a record so the operation can be reversed. This is Nestimate's counterpart of `tna::prune()`; the `net_` prefix avoids a name clash with `tna::prune()`.

Usage

```
net_prune(
  x,
  method = "threshold",
  threshold = 0.1,
  lowest = 0.05,
  level = 0.5,
  boot = NULL,
  ...
)

## S3 method for class 'netobject'
net_prune(
  x,
  method = "threshold",
  threshold = 0.1,
  lowest = 0.05,
  level = 0.5,
  boot = NULL,
  ...
)

## S3 method for class 'netobject_group'
net_prune(
  x,
  method = "threshold",
  threshold = 0.1,
  lowest = 0.05,
  level = 0.5,
  boot = NULL,
  ...
)

## Default S3 method:
net_prune(
  x,
  method = "threshold",
```

```

    threshold = 0.1,
    lowest = 0.05,
    level = 0.5,
    boot = NULL,
    ...
  )

```

Arguments

x	A netobject or netobject_group.
method	One of "threshold", "lowest", "disparity", "bootstrap". Default "threshold".
threshold	Numeric cut-off for method = "threshold". Default 0.1.
lowest	Quantile (0-1) for method = "lowest". Default 0.05.
level	Significance level (0-1) for method = "disparity". Default 0.5.
boot	Optional precomputed net_bootstrap for method = "bootstrap".
...	Passed to bootstrap_network when method = "bootstrap".

Details

Pruning is non-destructive: the pruned network carries a "pruning" attribute holding the original weights, the pruned weights, the parameters used, and a tidy table of removed edges. Use [net_deprune](#) to restore the original weights and [net_reprune](#) to re-apply the pruning, both without recomputation. [net_pruning_details](#) reports what was removed.

Methods:

"threshold" Remove edges with weight $\leq$ threshold.

"lowest" Remove the lowest lowest quantile of non-zero edges.

"disparity" Serrano disparity-filter backbone at significance level.

"bootstrap" Remove edges deemed non-significant by [bootstrap_network](#) (pass a precomputed result via boot, or extra bootstrap arguments via ...).

For "threshold", "lowest", and "disparity" an edge is dropped only when its removal leaves the network weakly connected.

Diagonal self-loops (self-transitions) are observed data: they are counted equally when computing the cut-off but are never removed by any method. (This is a deliberate divergence from `tna::prune()`, which prunes self-loops like any other edge.)

Value

The input network (or group) with pruned \$weights and a "pruning" attribute. Class is unchanged.

See Also

[net_deprune](#), [net_reprune](#), [net_pruning_details](#)

Examples

```
seqs <- data.frame(
  V1 = c("A", "B", "A", "C", "B"), V2 = c("B", "C", "B", "A", "C"),
  V3 = c("C", "A", "C", "B", "A"))
net <- build_network(seqs, method = "relative")
pruned <- net_prune(net, method = "threshold", threshold = 0.2)
net_pruning_details(pruned)
```

net_pruning_details	<i>Report Network Pruning Details</i>
---------------------	---------------------------------------

Description

Returns the edges removed by [net_prune](#) as a tidy one-row-per-edge data frame, with the method, cut-off, and retained/removed counts attached as attributes and shown by its print method.

Usage

```
net_pruning_details(x, ...)

## S3 method for class 'netobject'
net_pruning_details(x, ...)

## S3 method for class 'netobject_group'
net_pruning_details(x, ...)

## Default S3 method:
net_pruning_details(x, ...)
```

Arguments

x	A pruned netobject or netobject_group.
...	Ignored.

Value

For a netobject: a net_pruning_details data frame (columns from, to, weight) of removed edges. For a netobject_group: a named list of such data frames.

See Also

[net_prune](#)

Examples

```
seqs <- data.frame(
  V1 = c("A", "B", "A", "C", "B"), V2 = c("B", "C", "B", "A", "C"),
  V3 = c("C", "A", "C", "B", "A"))
net <- build_network(seqs, method = "relative")
net_pruning_details(net_prune(net, threshold = 0.2))
```

net_reprune

*Re-apply Network Pruning***Description**

Re-applies a previously computed pruning that was undone by [net_deprune](#), without recomputation.

Usage

```
net_reprune(x, ...)
```

```
## S3 method for class 'netobject'
```

```
net_reprune(x, ...)
```

```
## S3 method for class 'netobject_group'
```

```
net_reprune(x, ...)
```

```
## Default S3 method:
```

```
net_reprune(x, ...)
```

Arguments

x A depruned netobject or netobject_group.

... Ignored.

Value

The network (or group) with pruned weights re-applied and its pruning marked active.

See Also

[net_prune](#), [net_deprune](#)

Examples

```
seqs <- data.frame(V1 = c("A", "B", "C"), V2 = c("B", "C", "A"),
  V3 = c("C", "A", "B"))
net <- build_network(seqs, method = "relative")
pruned <- net_prune(net, threshold = 0.2)
undone <- net_deprune(pruned)
net_reprune(undone)
```

passage_time	<i>Mean First Passage Times</i>
--------------	---------------------------------

Description

Computes the full matrix of mean first passage times (MFPT) for a Markov chain. Element M_{ij} is the expected number of steps to travel from state i to state j for the first time. The diagonal equals the mean recurrence time $1/\pi_i$.

Usage

```
passage_time(x, states = NULL, normalize = TRUE)

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

## S3 method for class 'net_mpt'
plot(
  x,
  log_scale = TRUE,
  digits = 1,
  title = "Mean First Passage Times",
  low = "#004d00",
  high = "#ccffcc",
  ...
)
```

Arguments

x	A netobject, cograph_network, tna object, row-stochastic numeric transition matrix, or a wide sequence data.frame (rows = actors, columns = time-steps; a relative transition network is built automatically).
states	Character vector. Restrict output to these states. NULL (default) keeps all states.
normalize	Logical. If TRUE (default), rows that do not sum to 1 are normalized automatically (with a warning).
object	A net_mpt object (for summary).
...	Ignored.
log_scale	Logical. Apply log transform to the fill scale for better contrast? Default TRUE.
digits	Integer. Decimal places displayed in cells. Default 1.
title	Character. Plot title.
low	Character. Hex colour for the low end (short passage time). Default dark green "#004d00".
high	Character. Hex colour for the high end (long passage time). Default pale green "#ccffcc".

Details

Uses the Kemeny-Snell fundamental matrix formula:

$$M_{ij} = \frac{Z_{jj} - Z_{ij}}{\pi_j}, \quad Z = (I - P + \Pi)^{-1}$$

where $\Pi_{ij} = \pi_j$. Requires an ergodic (irreducible, aperiodic) chain.

Value

An object of class "net_mpt" with:

matrix Full $n \times n$ MFPT matrix. Row i , column j = expected steps from state i to state j . Diagonal = mean recurrence time $1/\pi_i$.

stationary Named numeric vector: stationary distribution π .

return_times Named numeric vector: $1/\pi_i$ per state.

states Character vector of state names.

summary.net_mpt returns a data frame with one row per state and columns state, return_time, stationary, mean_out (mean steps to other states), mean_in (mean steps from other states).

References

Kemeny, J.G. and Snell, J.L. (1976). *Finite Markov Chains*. Springer-Verlag.

See Also

[markov_stability](#), [build_network](#)

Examples

```
net <- build_network(as.data.frame(trjectories), method = "relative")
pt <- passage_time(net)
print(pt)

plot(pt)
```

Description

Extracts higher-order pathway strings suitable for `cograph::plot_simplicial()`. Each pathway represents a multi-step dependency: source states lead to a target state.

For `net_hon`: extracts edges where the source node is higher-order (order > 1), i.e., the transitions that differ from first-order Markov.

For `net_hypa`: extracts anomalous paths (over- or under-represented relative to the hypergeometric null model).

For `net_mogen`: extracts all transitions at the optimal order (or a specified order).

Usage

```
pathways(x, ...)

## S3 method for class 'net_hon'
pathways(x, min_count = 1L, min_prob = 0, top = NULL, order = NULL, ...)

## S3 method for class 'net_hypa'
pathways(x, type = "all", ...)

## S3 method for class 'netobject'
pathways(x, ho_method = c("hon", "hypa"), ...)

## S3 method for class 'net_association_rules'
pathways(x, top = NULL, min_lift = NULL, min_confidence = NULL, ...)

## S3 method for class 'net_link_prediction'
pathways(x, method = NULL, top = 10L, evidence = TRUE, max_evidence = 3L, ...)

## S3 method for class 'net_mogen'
pathways(x, order = NULL, min_count = 1L, min_prob = 0, top = NULL, ...)
```

Arguments

<code>x</code>	A higher-order network object (<code>net_hon</code> , <code>net_hypa</code> , or <code>net_mogen</code>).
<code>...</code>	Additional arguments.
<code>min_count</code>	Integer. Minimum transition count to include (default: 1).
<code>min_prob</code>	Numeric. Minimum transition probability to include (default: 0).
<code>top</code>	Integer or NULL. Return only the top N pathways ranked by count (default: NULL = all).
<code>order</code>	Integer or NULL. Markov order to extract. Default: optimal order from model selection.
<code>type</code>	Character. Which anomalies to include: "all" (default), "over", or "under".
<code>ho_method</code>	Character. Higher-order method: "hon" (default) or "hypa".
<code>min_lift</code>	Numeric or NULL. Additional lift filter applied on top of the object's original threshold (default: NULL).


```
seqs <- data.frame(
  V1 = sample(LETTERS[1:5], 50, TRUE),
  V2 = sample(LETTERS[1:5], 50, TRUE),
  V3 = sample(LETTERS[1:5], 50, TRUE)
)
net <- build_network(seqs, method = "relative")
pred <- predict_links(net, methods = "common_neighbors")
pathways(pred)
```

path_counts

Count Path Frequencies in Trajectory Data

Description

Counts the frequency of k-step paths (k-grams) across all trajectories. Useful for understanding which sequences dominate the data before applying formal models.

Usage

```
path_counts(data, k = 2L, top = NULL)
```

Arguments

data	A list of character vectors (trajectories) or a data.frame (rows = trajectories, columns = time points).
k	Integer. Length of the path / n-gram (default 2). A k of 2 counts individual transitions; k of 3 counts two-step paths, etc. Must be a whole number; a non-integer value is an error rather than being silently truncated.
top	Integer or NULL. If set, returns only the top N most frequent paths (default NULL = all).

Value

A data frame with columns: path, count, proportion.

Examples

```
trajs <- list(c("A", "B", "C", "D"), c("A", "B", "D", "C"))
path_counts(trajs, k = 2)
```

```
path_counts(trajs, k = 3, top = 10)
```

path_dependence	<i>Per-Context Path Dependence at Order k</i>
-----------------	---

Description

Diagnoses where a chain's order-1 Markov assumption fails by comparing, for each order-k context $(s_1, \dots, s_{k-1})$, the empirical next-state distribution $P(s_k \mid s_1, \dots, s_{k-1})$ against the order-1 prediction $P(s_k \mid s_{k-1})$ that uses only the most recent state. Returns a tidy per-context table sorted by Kullback-Leibler divergence so the analyst can see exactly *which* histories carry extra predictive information.

Usage

```
path_dependence(x, order = 2L, min_count = 5L, base = 2)
```

Arguments

x	A wide sequence data.frame / matrix (rows = actors, columns = time-steps), or a netobject that carries the source data.
order	Integer. Order of the conditioning context. order = 2 (default) compares 2-step memory against 1-step; order = 3 compares 3-step memory; etc. Must be a whole number; a non-integer value is an error rather than being silently truncated.
min_count	Integer. Drop contexts seen fewer than this many times. Default 5. Very small samples produce noisy KL estimates.
base	Numeric. Logarithm base for entropy and KL. Default 2 (bits).

Details

For each context $c = (s_1, \dots, s_{k-1})$ occurring at least min_count times, the function computes:

- the empirical conditional $P_k(\cdot \mid c)$ from k-gram counts;
- the order-1 prediction $P_1(\cdot \mid s_{k-1})$ from the most recent state alone (the bigram-marginal estimator);
- the entropy drop $H(P_1) - H(P_k)$ - bits of uncertainty removed by extending memory by one step in this specific context;
- the Kullback-Leibler divergence $D_{KL}(P_k \parallel P_1)$ - bits of "surprise" if you used the order-1 model when the order-k model is true.

KL = 0 means longer history adds no information for that context. H_drop > 0 means longer history sharpens the prediction; H_drop < 0 indicates the order-k context happens to spread probability across more outcomes than order-1 alone (small-sample noise or genuine context-induced uncertainty - inspect n).

Contexts where flips = TRUE are the substantively interesting ones: the longer history changes the *modal* prediction, not just its confidence.

Pair this with [markov_order_test](#) (which decides whether order-k is needed *globally*) to see the chain-level decision broken down per context.

Value

An object of class "net_path_dependence" with

contexts tidy data.frame, one row per order-k context, sorted by KL descending. Columns: context (e.g. "A -> B"), n (count), H_order1 (entropy of $P(\cdot \mid s_{k-1})$), H_orderk (entropy of $P(\cdot \mid \text{context})$), H_drop (= H_order1 - H_orderk), KL (= $D_{KL}(P_k \parallel P_1)$), top_o1 (most likely next state under order-1), top_ok (most likely next state under order-k), flips (logical: did the most likely next state change?).

chain list with chain-level summaries: KL_weighted (count-weighted mean KL across contexts), H_drop_weighted (count-weighted mean entropy drop), n_contexts, n_flips (contexts where the most-likely next state changed).

order integer

base numeric

min_count integer

states character vector

References

Cover, T.M. & Thomas, J.A. (2006). *Elements of Information Theory*, 2nd ed., chapters 2 and 4. Wiley. (KL divergence and conditional entropy.)

See Also

[markov_order_test](#), [transition_entropy](#), [build_mogen](#)

Examples

```
data(trajectories, package = "Nestimate")
pd <- path_dependence(as.data.frame(trajectories), order = 2)
print(pd)
summary(pd)
plot(pd)
```

permutation

Permutation Test for Network Comparison

Description

Compares two networks estimated by [build_network](#) using a permutation test. Works with all built-in methods (transition and association) as well as custom registered estimators. The test shuffles which observations belong to which group, re-estimates networks, and tests whether observed edge-wise differences exceed chance.

For transition methods ("relative", "frequency", "co_occurrence"), uses a fast pre-computation strategy: per-sequence count matrices are computed once, and each permutation iteration only shuffles group labels and computes group-wise colSums.

For association methods ("cor", "pcor", "glasso", and custom estimators), the full estimator is called on each permuted group split.

If either transition network contains only one sequence, the function warns that such a network is not recommended for permutation or other confirmatory testing.

permutation() also accepts two [net_edge_betweenness](#) objects. In that case it permutes the source networks, recomputes edge betweenness for each shuffled split, and tests edge-betweenness differences. The two edge-betweenness objects must come from the same source method and use the same invert setting.

Usage

```
permutation(
  x,
  y = NULL,
  iter = 1000L,
  alpha = 0.05,
  paired = FALSE,
  adjust = "none",
  measures = NULL,
  nlambda = 50L,
  seed = NULL
)
```

Arguments

x	A netobject (from build_network) or a net_edge_betweenness object.
y	A netobject (from build_network) or a net_edge_betweenness object. Must use the same method and have the same nodes as x.
iter	Integer. Number of permutation iterations (default: 1000).
alpha	Numeric. Significance level (default: 0.05).
paired	Logical. If TRUE, permute within pairs (requires equal number of observations in x and y). Default: FALSE.
adjust	Character. p-value adjustment method passed to p.adjust (default: "none"). Common choices: "holm", "BH", "bonferroni".
measures	Character vector of centrality measures to permutation-test in addition to the edges, or "all" for every built-in measure. Default NULL (edges only). When supplied, the result gains a \$centralities block matching the layout of <code>tna::permutation_test(measures)</code> : per state and measure it reports the observed difference, an effect size (difference / SD of the permutation null), and a permutation p-value, all using the same permuted networks as the edge test. Not supported for net_edge_betweenness inputs.
nlambda	Integer. Number of lambda values for the glassopath regularisation path (only used when method = "glasso"). Higher values give finer lambda resolution at the cost of speed. Default: 50.
seed	Integer or NULL. RNG seed for reproducibility.

Value

An object of class "net_permutation" containing:

x The first netobject.

y The second netobject.

diff Observed difference matrix ($x - y$).

diff_sig Observed difference where $p < \alpha$, else 0.

p_values P-value matrix (adjusted if `adjust != "none"`).

effect_size Effect size matrix (observed diff / SD of permutation diffs).

summary Long-format data frame of edge-level results.

method The network estimation method.

source_method For edge-betweenness tests, the source network method.

iter Number of permutation iterations.

alpha Significance level used.

paired Whether paired permutation was used.

adjust p-value adjustment method used.

centralities Present only when `measures` is supplied. A list with stats (one row per state-by-measure: `state`, `centrality`, `diff_true`, `effect_size`, `p_value`), `diffs_true` (wide observed differences), and `diffs_sig` (observed differences where $p < \alpha$, else 0).

See Also

[bayes_compare](#) for the Bayesian complement: instead of "is this difference more extreme than chance?" it answers "how probable is a difference, and how large?"; [build_network](#), [bootstrap_network](#), [print.net_permutation](#), [summary.net_permutation](#)

Examples

```
s1 <- data.frame(V1 = c("A", "B", "C"), V2 = c("B", "C", "A"))
s2 <- data.frame(V1 = c("A", "C", "B"), V2 = c("C", "B", "A"))
n1 <- build_network(s1, method = "relative")
n2 <- build_network(s2, method = "relative")
perm <- permutation(n1, n2, iter = 10)

set.seed(1)
d1 <- data.frame(V1 = sample(LETTERS[1:4], 20, TRUE),
                 V2 = sample(LETTERS[1:4], 20, TRUE),
                 V3 = sample(LETTERS[1:4], 20, TRUE))
d2 <- data.frame(V1 = sample(LETTERS[1:4], 20, TRUE),
                 V2 = sample(LETTERS[1:4], 20, TRUE),
                 V3 = sample(LETTERS[1:4], 20, TRUE))
net1 <- build_network(d1, method = "relative")
net2 <- build_network(d2, method = "relative")
perm <- permutation(net1, net2, iter = 100, seed = 42)
print(perm)
summary(perm)
```

persistence_landscape *Persistence Landscape*

Description

Computes the persistence landscape (Bubenik 2015) from a persistence diagram. Each (birth, death) pair contributes a tent function

$$\Lambda_{(b,d)}(t) = \max(0, \min(t - b, d - t)).$$

The k -th landscape function $\lambda^{(k)}(t)$ is the k -th largest of $\{\Lambda_{(b_i,d_i)}(t)\}_i$ at each t . Landscapes are stable under bottleneck distance and form a Banach-space embedding of persistence diagrams.

Usage

```
persistence_landscape(ph, k_max = 5L, dimension = 1L, t_grid = NULL)
```

Arguments

ph	A persistent_homology object or a data.frame with columns dimension, birth, death.
k_max	Maximum landscape index to compute (default 5). Must be a single positive integer.
dimension	Integer scalar – which homology dimension to compute the landscape for. Default 1.
t_grid	Numeric vector of evaluation points. NULL (default) uses an even grid of 200 points covering the union of pair intervals.

Value

A persistence_landscape object with:

landscape Data frame: k, t, value.

dimension Integer scalar.

k_max Integer scalar.

t_grid Numeric vector.

References

Bubenik, P. (2015). Statistical topological data analysis using persistence landscapes. *Journal of Machine Learning Research* **16**, 77-102.

Examples

```
mat <- matrix(c(0, .6, .5, .6, 0, .4, .5, .4, 0), 3, 3)
rownames(mat) <- colnames(mat) <- c("A", "B", "C")
ph <- persistent_homology(mat, n_steps = 5)
pl <- persistence_landscape(ph, k_max = 3, dimension = 0)
```

persistent_homology *Persistent Homology*

Description

Computes persistent homology via full boundary-matrix reduction over $\mathbb{Z}/2$ (Edelsbrunner, Letscher & Zomorodian 2000). The returned persistence diagram pairs each k-dimensional homology class to the simplex whose addition creates it (birth) and the simplex whose addition destroys it (death). Essential classes - those never killed - are reported with death = $\emptyset$ in clique mode (similarity scale, descending) and death = Inf in VR mode (distance scale, ascending).

Two filtration modes are supported:

type = "clique" Weighted clique filtration. Input is treated as a similarity matrix; high-weight simplices appear early. For each k-simplex σ , the filtration value is $\min_{(i,j) \in \sigma} |w(i,j)|$. Thresholds run high to low.

type = "vr" Vietoris-Rips filtration on a non-negative distance matrix. For each k-simplex σ , the filtration value is $\max_{(i,j) \in \sigma} d(i,j)$. Thresholds run low to high. Use max_scale to cap the filtration diameter.

Usage

```
persistent_homology(
  x,
  n_steps = 20L,
  max_dim = 3L,
  type = "clique",
  max_scale = NULL
)
```

Arguments

x	A square matrix, tna, or netobject. For type = "vr", must be a non-negative distance matrix.
n_steps	Number of grid points for the reported Betti curve (default 20). The persistence diagram itself is exact - it does not depend on n_steps.
max_dim	Maximum simplex dimension to track (default 3).
type	Filtration: "clique" (default, similarity-weighted) or "vr" (Vietoris-Rips on distances).
max_scale	For type = "vr" only: cap on edge length. Edges with $d(i,j) > \text{max_scale}$ are excluded. NULL (default) uses $\max(d)$.

Value

A persistent_homology object with:

betti_curve Data frame: threshold, dimension, betti.

persistence Data frame of birth-death pairs: dimension, birth, death, persistence. Sorted by descending persistence.

thresholds Numeric vector of grid thresholds.

mode Either "clique" or "vr".

References

Edelsbrunner, H., Letscher, D., & Zomorodian, A. (2000). Topological persistence and simplification. *Discrete & Computational Geometry* **28**, 511-533.

Examples

```
mat <- matrix(c(0,.6,.5,.6,0,.4,.5,.4,0), 3, 3)
colnames(mat) <- rownames(mat) <- c("A","B","C")
ph <- persistent_homology(mat, n_steps = 10)
print(ph)
```

plot.boot_glasso	<i>Plot Method for boot_glasso</i>
------------------	------------------------------------

Description

Plots bootstrap results for GLASSO networks.

Usage

```
## S3 method for class 'boot_glasso'
plot(x, type = "edges", measure = NULL, ...)
```

Arguments

x	A boot_glasso object.
type	Character. Plot type: "edges" (default), "stability", "edge_diff", "centrality_diff", or "inclusion".
measure	Character. Centrality measure for type = "centrality_diff" (default: first available measure).
...	Additional arguments passed to plotting functions. For type = "edge_diff" and type = "centrality_diff", accepts order: "sample" (default, sorted by value) or "id" (alphabetical).

Value

A ggplot object, invisibly.

Examples

```
set.seed(1)
dat <- as.data.frame(matrix(rnorm(60), ncol = 3))
bg <- boot_glasso(dat, iter = 10, cs_iter = 5, centrality = "strength")
plot(bg, type = "edges")

set.seed(42)
mat <- matrix(rnorm(60), ncol = 4)
colnames(mat) <- LETTERS[1:4]
boot <- boot_glasso(as.data.frame(mat), iter = 20, cs_iter = 10,
  centrality = "strength", seed = 42)
plot(boot, type = "edges")
```

plot.chain_structure *Plot method for chain_structure*

Description

Renders the hitting-probability matrix as a heatmap, with rows and columns ordered by communicating class so the block structure is visible at a glance. State labels along both axes are coloured by classification (absorbing / recurrent / transient). The subtitle summarises the chain-level properties (regular, reversible).

Usage

```
## S3 method for class 'chain_structure'
plot(x, show_values = TRUE, digits = 2L, ...)
```

Arguments

<code>x</code>	A chain_structure object.
<code>show_values</code>	Logical. If TRUE (default), prints the numeric probability inside each cell. Set FALSE for large state spaces ($n > 10$) where labels overlap.
<code>digits</code>	Integer. Decimal places for in-cell labels.
<code>...</code>	Ignored.

Details

Cell colour encodes $P(\text{ever reach } j \mid \text{start at } i)$. The diagonal uses the return-time convention ($P(\text{return to } j \text{ in } \geq 1 \text{ steps})$), matching `markovchain::hittingProbabilities`. A non-irreducible chain shows zero off-block entries – visual evidence of one-way doors between behavioural phases. An absorbing chain shows a column of 1's for the absorbing state.

Value

A ggplot object.

plot.cluster_choice *Plot Method for cluster_choice*

Description

Six explicit chart types plus a smart "auto" default. The user picks the shape; the function does not editorialise (no "best" annotation, no interpretive subtitles, no inferred recommendation).

Usage

```
## S3 method for class 'cluster_choice'
plot(
  x,
  type = c("auto", "lines", "bars", "heatmap", "tradeoff", "facet"),
  abbrev = FALSE,
  combined = TRUE,
  ...
)
```

Arguments

x	A cluster_choice object.
type	Character. One of "auto" (default), "lines", "bars", "heatmap", "tradeoff", "facet".
abbrev	Logical. If TRUE, dissimilarity and method names shown on tick labels and point labels are shortened (e.g. "hamming" -> "ham", "ward.D2" -> "wD2"). The legend shows the full canonical name. Default FALSE.
combined	Only meaningful for type = "facet". When TRUE (default), all methods are shown in one ggplot via facet_wrap(~ method). When FALSE, returns a named list of single-panel ggplots, one per method.
...	Unsupported. Supplying unused arguments raises an error.

Details

Type cheat-sheet:

"auto" Default. Picks one of the others based on which axes were swept. k-only -> "lines"; one categorical axis swept -> "bars"; k plus one categorical -> "lines"; k plus two categoricals -> "facet"; both categoricals without k -> "heatmap".

"lines" Silhouette across k (and mean_within_dist when k is the only swept axis), one line per non-k axis when present.

"bars" Horizontal bar chart of silhouette per axis level. Bars sorted by silhouette.

"heatmap" Tiled silhouette across two categorical axes. Requires both dissimilarity and method swept.

"tradeoff" Scatter: silhouette (y) vs size_ratio (x). Works for any sweep; labels each point.

"facet" Lines vs k, colour by one categorical axis, facet by another. Requires k plus two categorical.

Asking for a type the data can't support raises an error pointing at the alternatives.

Value

A ggplot object, invisibly; for type = "facet" with combined = FALSE, a named list of ggplots.

<code>plot.mcml_pc</code>	<i>Plot an MCML-PC Object</i>
---------------------------	-------------------------------

Description

Heatmap of the macro (cluster-level) weights with the package's diverging palette. For the two-layer network rendering use `cograph::plot_mcml()`, which accepts `mcml_pc` objects and draws them undirected.

Usage

```
## S3 method for class 'mcml_pc'
plot(x, digits = 2, ...)
```

Arguments

- `x` An `mcml_pc` object.
- `digits` Number of digits for tile labels (default 2).
- `...` Additional arguments (ignored).

Value

A ggplot object.

<code>plot.mmm_compare</code>	<i>Plot Method for mmm_compare</i>
-------------------------------	------------------------------------

Description

Plot Method for `mmm_compare`

Usage

```
## S3 method for class 'mmm_compare'
plot(x, ...)
```

Arguments

`x` An `mmm_compare` object.

`...` Unsupported. Supplying unused arguments raises an error.

Value

A `ggplot` object, invisibly.

Examples

```
seqs <- data.frame(V1 = sample(c("A", "B", "C"), 30, TRUE),
                   V2 = sample(c("A", "B", "C"), 30, TRUE))
cmp <- compare_mmm(seqs, k = 2:3, n_starts = 1, max_iter = 10, seed = 1)
plot(cmp)

set.seed(1)
seqs <- data.frame(
  V1 = sample(c("A", "B", "C"), 30, TRUE),
  V2 = sample(c("A", "B", "C"), 30, TRUE),
  V3 = sample(c("A", "B", "C"), 30, TRUE)
)
cmp <- compare_mmm(seqs, k = 2:3, n_starts = 5, seed = 1)
plot(cmp)
```

`plot.mosaic_analysis` *Plot method for mosaic_analysis objects*

Description

Re-renders the flat mosaic from the stored contingency table and residuals, so styling can be changed without re-running the test. Any flat-mosaic styling argument (`tile_label`, `pct_base`, `col_label_side`, `legend_size`, ...) may be overridden via

Usage

```
## S3 method for class 'mosaic_analysis'
plot(x, ...)
```

Arguments

`x` A `mosaic_analysis` object.

`...` Styling overrides forwarded to the flat renderer.

Value

The `ggplot` object, invisibly (drawn as a side effect).

Examples

```
df <- data.frame(
  a = sample(c("X", "Y", "Z"), 200, replace = TRUE),
  b = sample(c("P", "Q"), 200, replace = TRUE)
)
res <- mosaic_analysis(df, "a", "b", min_count = 5)

plot(res, tile_label = "percent", legend_position = "bottom")
```

plot.net_association_rules

Plot Method for net_association_rules

Description

Scatter plot of association rules: support vs confidence, with point size proportional to lift.

Usage

```
## S3 method for class 'net_association_rules'
plot(x, ...)
```

Arguments

x A net_association_rules object.

... Additional arguments passed to ggplot2 functions.

Value

A ggplot object, invisibly.

Examples

```
trans <- list(c("A", "B", "C"), c("A", "B"), c("B", "C", "D"),
             c("A", "C", "D"), c("A", "B", "D"), c("B", "C"))
rules <- association_rules(trans, min_support = 0.3, min_confidence = 0.3,
                          min_lift = 0)
plot(rules)
```

plot.net_bayes	<i>Plot method for net_bayes</i>
----------------	----------------------------------

Description

Draws a differential transition network as a directed chord diagram. Edge colour encodes the signed posterior mean difference (x stronger vs y stronger) and edge width its magnitude.

Usage

```
## S3 method for class 'net_bayes'
plot(x, significant_only = TRUE, title = NULL, ...)
```

Arguments

x	A net_bayes object.
significant_only	Logical. Show only credibly-different edges (default TRUE).
title	Optional plot title.
...	Additional arguments passed to <code>cograph::splot()</code> when <code>cograph</code> is available.

Value

Invisibly, the `cograph` network returned by `cograph::splot()` when `cograph` is available; otherwise a fallback `ggplot` object.

Examples

```
s1 <- data.frame(V1 = c("A", "B", "C"), V2 = c("B", "C", "A"))
s2 <- data.frame(V1 = c("A", "C", "B"), V2 = c("C", "B", "A"))
b <- bayes_compare(build_network(s1, method = "relative"),
                   build_network(s2, method = "relative"),
                   draws = 500, seed = 1)
plot(b, significant_only = FALSE)
```

plot.net_centrality	<i>Plot centrality measures</i>
---------------------	---------------------------------

Description

Plot centrality measures

Usage

```
## S3 method for class 'net_centrality'
plot(
  x,
  reorder = TRUE,
  ncol = 3L,
  type = c("bar", "line", "heatmap"),
  scales = c("free_x", "fixed"),
  profile_scale = c("measure", "none"),
  labels = TRUE,
  drop_zero = FALSE,
  ...
)
```

Arguments

x	A net_centrality object returned by net_centrality .
reorder	Logical. Reorder states within each centrality panel by centrality value. Default: TRUE.
ncol	Integer. Number of facet columns.
type	Plot type. "bar" shows one faceted horizontal bar chart per measure; "line" shows state profiles as lines across measures (the value "profile" is still accepted as an alias); "heatmap" shows a states-by-measures tile grid, each measure scaled to 0–1 for cross-measure comparability with the raw value printed in the tile.
scales	Facet scale mode. "free_x" uses free centrality axes; "fixed" keeps a common centrality axis.
profile_scale	Scaling used by type = "line". "measure" rescales each centrality measure to 0–1 before drawing cross-measure profiles; "none" uses raw values.
labels	Logical. Add compact value labels.
drop_zero	Logical. Drop measures whose values are all (near) zero so empty panels do not waste space. Default: FALSE (every requested measure is shown).
...	Additional arguments ignored.

Value

A ggplot object.

plot.net_centrality_group
<i>Plot grouped centrality measures</i>

Description

Plot grouped centrality measures

Usage

```
## S3 method for class 'net_centrality_group'
plot(
  x,
  reorder = TRUE,
  ncol = 3L,
  type = c("bar", "line", "delta"),
  scales = c("free_x", "fixed"),
  palette = "Set2",
  profile_scale = c("measure", "none"),
  labels = FALSE,
  drop_zero = FALSE,
  ...
)
```

Arguments

<code>x</code>	A <code>net_centrality_group</code> object returned by net_centrality .
<code>reorder</code>	Logical. Reorder states by their mean value within each centrality panel. Default: TRUE.
<code>ncol</code>	Integer. Number of facet columns.
<code>type</code>	Plot type. "bar" shows grouped bars within each measure; "line" facets by state and draws one line per group across centrality measures ("profile" is accepted as an alias); "delta" draws a diverging bar of group differences. With two groups it is the per-state difference (second group minus first); with three or more groups it is each group's deviation from the per-state group mean, so the largest gaps stand out either way.
<code>scales</code>	Facet scale mode. "free_x" uses free centrality axes; "fixed" keeps a common centrality axis.
<code>palette</code>	Brewer palette for groups.
<code>profile_scale</code>	Scaling used by <code>type = "line"</code> . "measure" rescales each centrality measure to 0–1 before drawing cross-measure profiles; "none" uses raw values.
<code>labels</code>	Logical. Add compact value labels.
<code>drop_zero</code>	Logical. Drop measures whose values are all (near) zero so empty panels do not waste space. Default: FALSE.
<code>...</code>	Additional arguments ignored.

Value

A ggplot object.

plot.net_clustering *Plot Sequence Clustering Results*

Description

Plot Sequence Clustering Results

Usage

```
## S3 method for class 'net_clustering'
plot(
  x,
  type = c("silhouette", "mds", "heatmap", "predictors"),
  combined = TRUE,
  ...
)
```

Arguments

<code>x</code>	A <code>net_clustering</code> object.
<code>type</code>	Character. Plot type: "silhouette" (per-observation silhouette bars), "mds" (2D MDS projection), or "heatmap" (distance matrix heatmap ordered by cluster). Default: "silhouette".
<code>combined</code>	Logical. For type = "predictors" only: when TRUE (default), covariate forest panels are combined into a single faceted plot; when FALSE, a list of separate ggplots is returned.
<code>...</code>	Unsupported. Supplying unused arguments raises an error.

Value

A ggplot object (invisibly).

Examples

```
seqs <- data.frame(V1 = c("A", "B", "C", "A", "B"), V2 = c("B", "C", "A", "B", "A"),
                  V3 = c("C", "A", "B", "C", "B"))
cl <- build_clusters(seqs, k = 2)
plot(cl, type = "silhouette")

set.seed(1)
seqs <- data.frame(
  V1 = sample(c("A", "B", "C"), 20, TRUE),
  V2 = sample(c("A", "B", "C"), 20, TRUE),
  V3 = sample(c("A", "B", "C"), 20, TRUE)
)
cl <- build_clusters(seqs, k = 2)
plot(cl, type = "silhouette")
```

plot.net_cluster_diagnostics

Plot Method for net_cluster_diagnostics

Description

Delegates to the original clustering object's plot method ([plot.net_clustering](#) for distance-based diagnostics, [plot.net_mmm_clustering](#) or [plot.net_mmm](#) for model-based). The diagnostics object itself stores no plot geometry – it just keeps a reference to the source so the existing visual layer is reused.

Usage

```
## S3 method for class 'net_cluster_diagnostics'
plot(x, type = NULL, ...)
```

Arguments

x	A net_cluster_diagnostics object.
type	Character. Forwarded to the underlying plot method. Valid values for distance: "silhouette" (default), "mds", "heatmap", "predictors". Valid values for mmm: "posterior" (default), "covariates" / "predictors".
...	Forwarded to the underlying plot method.

Value

A ggplot object, invisibly.

plot.net_comparison *Plot a network comparison*

Description

Visualises a net_comparison object. Currently supports the edge-weight scatterplot (default), with the diagonal reference (perfect agreement) and the OLS regression line annotated by Pearson, Spearman, and Kendall correlations.

Usage

```
## S3 method for class 'net_comparison'
plot(
  x,
  type = c("scatter", "heatmap", "diff_hist", "weight_dist", "all"),
  combined = TRUE,
  ...
)
```

Arguments

x	A net_comparison object from <code>compare_model()</code> .
type	Character. One of "scatter" (default - edge-weight scatter with OLS fit and correlation overlay), "heatmap" (n by n grid of x - y differences using the diverging palette), "diff_hist" (histogram of x - y absolute differences with rug + density), "weight_dist" (overlaid distributions of x and y edge weights), or "all" (2 by 2 grid of all four panels; requires the gridExtra package).
combined	When type = "all" and combined = TRUE (default), the four panels are stitched into a 2x2 gtable. When FALSE, returns a named list of the four ggplots so each can be printed, saved, or re-laid-out independently. Ignored for other type values.
...	Ignored.

Value

A ggplot object; for type = "all" with combined = TRUE a gtable arranged 2 by 2; for type = "all" with combined = FALSE a named list of four ggplots.

plot.net_edge_betweenness

Plot edge-betweenness scores

Description

Draws the edges of a `net_edge_betweenness` network ranked by their betweenness, as a horizontal bar chart. This is the tidy, cograph-free companion to the node-link diagram: render the diagram with `cograph::splot(eb)` and the ranking with `plot(eb)`.

Usage

```
## S3 method for class 'net_edge_betweenness'
plot(x, style = c("bar", "forest", "delta"), top_n = NULL, labels = TRUE, ...)
```

Arguments

x	A net_edge_betweenness network from <code>net_edge_betweenness</code> .
style	Plot style. "bar" (default) draws one horizontal bar per edge; "forest" draws a forest/lollipop chart (a stem from zero to a point) with a dashed reference line at the mean betweenness; "delta" draws each edge's deviation from the mean edge betweenness as a diverging bar (above the mean in blue, below in red).
top_n	Integer or NULL. Keep only the top_n highest edges. Default NULL (all edges with non-zero betweenness).
labels	Logical. Print the betweenness value beside each edge. Default TRUE.
...	Additional arguments (ignored).

Value

A ggplot object.

Examples

```
seqs <- data.frame(
  V1 = c("A", "B", "A", "C"), V2 = c("B", "C", "B", "A"),
  V3 = c("C", "A", "C", "B"))
eb <- net_edge_betweenness(build_network(seqs, method = "relative"))
plot(eb)
plot(eb, style = "forest")
plot(eb, style = "delta")
```

plot.net_gimme	<i>Plot Method for net_gimme</i>
----------------	----------------------------------

Description

Plot Method for net_gimme

Usage

```
## S3 method for class 'net_gimme'
plot(
  x,
  type = c("temporal", "contemporaneous", "individual", "counts", "fit"),
  subject = NULL,
  ...
)
```

Arguments

x	A net_gimme object.
type	Character. Plot type: "temporal", "contemporaneous", "individual", "counts", or "fit".
subject	Integer or character. Subject to plot for type = "individual".
...	Additional arguments (ignored).

Value

The input object, invisibly.

Examples

```

set.seed(1)
panel <- data.frame(
  id = rep(1:5, each = 20),
  t  = rep(seq_len(20), 5),
  A  = rnorm(100), B = rnorm(100), C = rnorm(100)
)
gm <- build_gimme(panel, vars = c("A","B","C"), id = "id", time = "t")
plot(gm, type = "temporal")

```

plot.net_honem

Plot Method for net_honem

Description

Plot Method for net_honem

Usage

```

## S3 method for class 'net_honem'
plot(x, dims = c(1L, 2L), ...)

```

Arguments

x	A net_honem object.
dims	Integer vector of length 2. Dimensions to plot (default: c(1, 2)).
...	Additional arguments passed to plot .

Value

The input object, invisibly.

Examples

```

seqs <- list(c("A","B","C","D"), c("A","B","C","A"), c("B","C","D","A"))
hem <- build_honem(build_hon(seqs, max_order = 2), dim = 2)
plot(hem)

```

```

seqs <- list(c("A","B","C","D"), c("A","B","C","A"), c("B","C","D","A"))
hon <- build_hon(seqs, max_order = 3)
he <- build_honem(hon, dim = 2)
plot(he)

```

plot.net_markov_order *Plot Method for net_markov_order*

Description

Two-panel professional visualization:

- Panel A: log-likelihood, AIC, BIC across tested orders with the selected order highlighted (both the permutation-selected order and the BIC-minimizing order are marked).
- Panel B: permutation null density per order with the observed G^2 as a vertical marker; colored by rejection at alpha.

Uses the Okabe-Ito colorblind-safe palette.

Usage

```
## S3 method for class 'net_markov_order'
plot(x, panel = c("both", "ic", "permutation"), combined = TRUE, ...)
```

Arguments

x	A net_markov_order object.
panel	Which panel(s) to render: "both", "ic", or "permutation". Default "both".
combined	When panel = "both" and combined = TRUE (default), the two panels are drawn side-by-side. If gridExtra is installed they are arranged into a single draw-able/saveable gtable (returned); otherwise base grid viewports draw both panels and a named list of the two ggplots is returned invisibly. When FALSE, returns that named list (ic, permutation) without drawing. Ignored when panel != "both".
...	Ignored.

Value

A ggplot (single panel); for panel = "both", either a gridExtra gtable (when **gridExtra** is installed) or a named list of two ggplots (ic, permutation) drawn side-by-side and returned invisibly.

Examples

```
# First-order Markov data: test should select order 1
set.seed(1)
states <- letters[1:4]
tm <- matrix(runif(16), 4, 4, dimnames = list(states, states))
tm <- tm / rowSums(tm)
seqs <- lapply(1:30, function(.) {
  s <- character(50); s[1] <- sample(states, 1)
  for (i in 2:50) s[i] <- sample(states, 1, prob = tm[s[i - 1], ])
  s
})
```

```
res <- markov_order_test(seqs, max_order = 3, n_perm = 300, seed = 1)
res$optimal_order
summary(res)
plot(res)
```

plot.net_mmm

Plot Method for net_mmm

Description

Plot Method for net_mmm

Usage

```
## S3 method for class 'net_mmm'
plot(x, type = c("posterior", "covariates"), combined = TRUE, ...)
```

Arguments

x	A net_mmm object.
type	Character. Plot type: "posterior" (default) or "covariates".
combined	Logical. For type = "covariates" only: when TRUE (default), covariate forest panels are combined into a single faceted plot; when FALSE, a list of separate ggplots is returned.
...	Unsupported. Supplying unused arguments raises an error.

Value

A ggplot object, invisibly.

Examples

```
seqs <- data.frame(V1 = sample(c("A","B","C"), 30, TRUE),
                  V2 = sample(c("A","B","C"), 30, TRUE))
mmm <- build_mmm(seqs, k = 2, n_starts = 1, max_iter = 10, seed = 1)
plot(mmm, type = "posterior")

set.seed(1)
seqs <- data.frame(
  V1 = sample(c("A","B","C"), 30, TRUE),
  V2 = sample(c("A","B","C"), 30, TRUE),
  V3 = sample(c("A","B","C"), 30, TRUE)
)
mmm <- build_mmm(seqs, k = 2, n_starts = 5, seed = 1)
plot(mmm, type = "posterior")
```

plot.net_mmm_clustering

Plot Method for MMM Clustering Attribute

Description

Plot routines for the MMM clustering metadata attached to a netobject_group by [cluster_mmm](#) (or [cluster_network](#) with cluster_by = "mmm"). Mirrors the type-driven surface of [plot.net_clustering](#) but covers only the metrics the EM fit produces – there is no distance matrix on an MMM clustering, so "silhouette" / "mds" / "heatmap" aren't defined here and the dispatcher raises a clear error if you ask for one of those on an MMM result.

Usage

```
## S3 method for class 'net_mmm_clustering'
plot(
  x,
  type = c("posterior", "covariates", "predictors"),
  combined = TRUE,
  ...
)
```

Arguments

x	A net_mmm_clustering object.
type	Character. One of "posterior" (default; histogram of max posterior probability per sequence, coloured by cluster), "covariates" or its alias "predictors" (covariate forest plot when cluster_mmm() was run with covariates).
combined	Logical. For type in "covariates" or "predictors" only: when TRUE (default), forest panels are combined into a single faceted plot; when FALSE, a list of separate ggplots is returned.
...	Unsupported. Supplying unused arguments raises an error.

Value

A ggplot object, invisibly.

Examples

```
seqs <- data.frame(V1 = sample(c("A","B","C"), 40, TRUE),
                   V2 = sample(c("A","B","C"), 40, TRUE))
grp <- cluster_mmm(seqs, k = 2, n_starts = 1, max_iter = 20, seed = 1)
plot(attr(grp, "clustering"), type = "posterior")
```

plot.net_mogen	<i>Plot Method for net_mogen</i>
----------------	----------------------------------

Description

Plot Method for net_mogen

Usage

```
## S3 method for class 'net_mogen'  
plot(x, type = c("ic", "likelihood"), ...)
```

Arguments

x	A net_mogen object.
type	Character. Plot type: "ic" (default) or "likelihood".
...	Additional arguments passed to plot .

Value

The input object, invisibly.

Examples

```
seqs <- list(c("A", "B", "C", "D"), c("A", "B", "C", "A"), c("B", "C", "D", "A"))  
mg <- build_mogen(seqs, max_order = 2)  
plot(mg)
```

```
seqs <- data.frame(  
  V1 = c("A", "B", "C", "A", "B"),  
  V2 = c("B", "C", "A", "B", "C"),  
  V3 = c("C", "A", "B", "C", "A")  
)  
mog <- build_mogen(seqs, max_order = 2L)  
plot(mog, type = "ic")
```

plot.net_path_dependence

Plot method for net_path_dependence

Description

Lollipop chart of per-context KL divergence, sorted descending. Point size is proportional to context count; points where the modal next state flips between orders are marked with an X to highlight substantively meaningful order-2 effects.

Usage

```
## S3 method for class 'net_path_dependence'
plot(x, top = 15L, title = NULL, ...)
```

Arguments

x	A net_path_dependence object.
top	Integer. Number of contexts to show (top by KL). Default 15.
title	Character. Plot title.
...	Ignored.

Value

A ggplot object.

plot.net_reliability *Plot Method for net_reliability*

Description

Density plots of split-half metrics faceted by metric type. Multi-model comparisons show overlaid densities colored by model.

Usage

```
## S3 method for class 'net_reliability'
plot(x, bins = 60L, combined = TRUE, ...)
```

Arguments

x	A net_reliability object.
bins	Integer. Number of histogram bins per panel (default 60).
combined	When TRUE (default), all four metrics are shown in one ggplot via facet_wrap(~metric). When FALSE, returns a named list of four single-panel ggplots, one per metric.
...	Additional arguments (ignored).

Value

A ggplot object (invisibly), or a named list of four ggplots when combined = FALSE.

Examples

```
net <- build_network(data.frame(V1 = c("A","B","C","A"),
  V2 = c("B","C","A","B")), method = "relative")
rel <- network_reliability(net, iter = 10)
plot(rel)

set.seed(1)
seqs <- data.frame(
  V1 = sample(c("A","B","C"), 30, TRUE),
  V2 = sample(c("A","B","C"), 30, TRUE),
  V3 = sample(c("A","B","C"), 30, TRUE)
)
net <- build_network(seqs, method = "relative")
rel <- network_reliability(net, iter = 20, seed = 1)
plot(rel)
```

plot.net_sequence_comparison

Plot Method for net_sequence_comparison

Description

Visualizes pattern-level standardized residuals across groups. Two styles are available:

"pyramid" Back-to-back bars of pattern proportions, shaded by each side's standardized residual.

Requires exactly 2 groups.

"heatmap" One tile per (pattern, group) cell, colored by standardized residual. Works for any number of groups.

Residuals are read directly from the resid_<group> columns in \$patterns, which are always populated regardless of the inference method chosen in sequence_compare.

Usage

```
## S3 method for class 'net_sequence_comparison'
plot(
  x,
  top_n = 10L,
  style = c("auto", "pyramid", "heatmap"),
  sort = c("statistic", "frequency"),
  alpha = 0.05,
  show_residuals = FALSE,
  ...
)
```

Arguments

<code>x</code>	A <code>net_sequence_comparison</code> object.
<code>top_n</code>	Integer. Show top N patterns. Default: 10.
<code>style</code>	Character. "auto" (default) draws the back-to-back pyramid for exactly 2 groups and the heatmap for any other number; "pyramid" and "heatmap" force a specific style.
<code>sort</code>	Character. "statistic" (default) ranks patterns by test statistic or residual magnitude. "frequency" ranks by total occurrence count across all groups.
<code>alpha</code>	Numeric. Significance threshold for p-value display in the pyramid: patterns with <code>p_value < alpha</code> are starred and drawn in bold dark text, the rest stay plain grey. Default: 0.05.
<code>show_residuals</code>	Logical. If TRUE, print the standardized residual value inside each pyramid bar. Default: FALSE. Ignored for the heatmap (which always shows residuals).
<code>...</code>	Additional arguments (ignored).

Value

A ggplot object, invisibly.

Examples

```
seqs <- data.frame(
  V1 = sample(LETTERS[1:4], 60, TRUE),
  V2 = sample(LETTERS[1:4], 60, TRUE),
  V3 = sample(LETTERS[1:4], 60, TRUE),
  V4 = sample(LETTERS[1:4], 60, TRUE)
)
grp <- rep(c("X", "Y"), 30)
net <- build_network(seqs, method = "relative")
res <- sequence_compare(net, group = grp, sub = 2:3, test = "chisq")
```

plot.net_stability	<i>Plot Method for net_stability</i>
--------------------	--------------------------------------

Description

Plots mean correlation vs drop proportion for each centrality measure. The CS-coefficient is marked where the curve crosses the threshold.

Usage

```
## S3 method for class 'net_stability'
plot(x, ...)
```

Arguments

`x` A `net_stability` object.
`...` Additional arguments (ignored).

Value

A `ggplot` object (invisibly).

Examples

```
net <- build_network(data.frame(V1 = c("A", "B", "C", "A"),
  V2 = c("B", "C", "A", "B")), method = "relative")
cs <- centrality_stability(net, iter = 10, drop_prop = 0.3)
plot(cs)

set.seed(1)
seqs <- data.frame(
  V1 = sample(c("A", "B", "C"), 30, TRUE),
  V2 = sample(c("A", "B", "C"), 30, TRUE),
  V3 = sample(c("A", "B", "C"), 30, TRUE)
)
net <- build_network(seqs, method = "relative")
stab <- centrality_stability(net, measures = c("InStrength", "OutStrength"),
  iter = 10)
plot(stab)
```

`plot.net_transition_entropy`

Plot method for net_transition_entropy

Description

Bar chart of per-state row entropy with overlaid horizontal lines at the entropy rate $h(P)$ (chain-level summary) and the maximum row entropy $\log_b n$ (uniform branching). Bar widths are proportional to the stationary probability so the visual area sums to the entropy rate.

Usage

```
## S3 method for class 'net_transition_entropy'
plot(x, title = "Transition Entropy", fill = "#0072B2", ...)
```

Arguments

`x` A `net_transition_entropy` object.
`title` Character. Plot title.
`fill` Character. Bar fill colour. Default Okabe-Ito blue.
`...` Ignored.

Value

A ggplot object.

```
plot.net_vertex_bootstrap
```

Plot Vertex Bootstrap Distributions

Description

Histogram of the bootstrap distribution per statistic, with the observed value (solid line) and confidence bounds (dashed lines).

Usage

```
## S3 method for class 'net_vertex_bootstrap'
plot(x, bins = 30, ...)
```

Arguments

x	A net_vertex_bootstrap object.
bins	Number of histogram bins (default 30).
...	Additional arguments (ignored).

Value

A ggplot object.

```
plot.net_vertex_comparison
```

Plot a Two-Network Vertex-Bootstrap Comparison

Description

Forest plot of the statistic differences with normal-approximation confidence intervals; differences whose interval excludes zero are the statistically distinguishable ones.

Usage

```
## S3 method for class 'net_vertex_comparison'
plot(x, ...)
```

Arguments

x	A net_vertex_comparison object.
...	Additional arguments (ignored).

Value

A ggplot object.

```
plot.pc_loading_stability
```

Plot Composite-Weight Stability

Description

Signed composite weights with bootstrap percentile intervals, faceted by cluster.

Usage

```
## S3 method for class 'pc_loading_stability'  
plot(x, ...)
```

Arguments

x	A pc_loading_stability object.
...	Additional arguments (ignored).

Value

A ggplot object.

```
plot.persistence_landscape
```

Plot Persistence Landscape

Description

Plot Persistence Landscape

Usage

```
## S3 method for class 'persistence_landscape'  
plot(x, ...)
```

Arguments

x	A persistence_landscape object.
...	Ignored.

Value

A ggplot.

Examples

```
mat <- matrix(c(0, .6, .5, .6, 0, .4, .5, .4, 0), 3, 3)
rownames(mat) <- colnames(mat) <- c("A", "B", "C")
ph <- persistent_homology(mat, n_steps = 5)
pl <- persistence_landscape(ph, k_max = 3, dimension = 0)
plot(pl)
```

plot.persistent_homology

Plot Persistent Homology

Description

Two panels: Betti curve (threshold vs Betti number) and persistence diagram (birth vs death). Persistence pairs come from full boundary- matrix reduction; essential classes are shown at the filtration boundary (death = 0 in clique mode, death = max_scale in VR mode).

Usage

```
## S3 method for class 'persistent_homology'
plot(x, combined = TRUE, ...)
```

Arguments

x	A persistent_homology object.
combined	When TRUE (default), the two panels are stitched side-by-side via <code>gridExtra::arrangeGrob</code> . When FALSE, returns a named list (betti_curve, persistence) of ggplots.
...	Ignored.

Value

A grid grob (invisibly) when combined = TRUE; a named list of two ggplots when combined = FALSE.

Examples

```
seqs <- data.frame(
  V1 = c("A", "B", "C", "A", "B"),
  V2 = c("B", "C", "A", "B", "C"),
  V3 = c("C", "A", "B", "C", "A")
)
net <- build_network(seqs, method = "relative")
ph <- persistent_homology(net)
if (requireNamespace("gridExtra", quietly = TRUE)) plot(ph)
```

plot.q_analysis	<i>Plot Q-Analysis</i>
-----------------	------------------------

Description

Two panels: Q-vector (components at each connectivity level) and structure vector (max simplex dimension per node).

Usage

```
## S3 method for class 'q_analysis'
plot(x, combined = TRUE, ...)
```

Arguments

x	A q_analysis object.
combined	When TRUE (default), the two panels are stitched side-by-side via <code>gridExtra::arrangeGrob</code> . When FALSE, returns a named list (<code>q_vector</code> , <code>structure_vector</code>) of ggplots.
...	Ignored.

Value

A grid grob (invisibly) when `combined = TRUE`; a named list of two ggplots when `combined = FALSE`.

Examples

```
seqs <- data.frame(
  V1 = c("A", "B", "C", "A", "B"),
  V2 = c("B", "C", "A", "B", "C"),
  V3 = c("C", "A", "B", "C", "A")
)
net <- build_network(seqs, method = "relative")
sc <- build_simplicial(net, type = "clique")
qa <- q_analysis(sc)
plot(qa)
```

plot.simplicial_complex

Plot a Simplicial Complex

Description

Produces a multi-panel summary: f-vector, simplicial degree ranking, and degree-by-dimension heatmap.

Usage

```
## S3 method for class 'simplicial_complex'
plot(x, combined = TRUE, ...)
```

Arguments

x	A simplicial_complex object.
combined	When TRUE (default), the four panels are stitched into a 2x2 gtable via <code>gridExtra::arrangeGrob</code> and drawn. When FALSE, returns a named list of the four ggplots (<code>f_vector</code> , <code>betti</code> , <code>degree</code> , <code>degree_heatmap</code>) so each can be printed, saved, or re-laid-out independently.
...	Ignored.

Value

A grid grob (invisibly) when `combined = TRUE`; a named list of four ggplots when `combined = FALSE`.

Examples

```
mat <- matrix(c(0,.6,.5,.6,0,.4,.5,.4,0), 3, 3)
colnames(mat) <- rownames(mat) <- c("A","B","C")
sc <- build_simplicial(mat, threshold = 0.3)
if (requireNamespace("gridExtra", quietly = TRUE)) plot(sc)
```

plot_mosaic

Draw a Marimekko / Mosaic Plot from a Tidy Data Frame

Description

Low-level rectangle-coordinate builder for marimekko (mosaic) plots. Column widths are proportional to the per-column total of weight; within each column, segments stack to height 1 with sub-heights proportional to each row's share of that column's total.

Usage

```
plot_mosaic(
  data,
  x,
  y,
  weight,
  fill = "y",
  colors = NULL,
  show_labels = TRUE,
  label_size = 3.5,
  x_label = NULL,
  y_label = NULL
)
```

Arguments

<code>data</code>	A data.frame in long form. Must contain the columns named in <code>x</code> , <code>y</code> , and <code>weight</code> .
<code>x</code>	Column name for the X (column) variable.
<code>y</code>	Column name for the Y (segment) variable.
<code>weight</code>	Column name for the cell weight (e.g. count).
<code>fill</code>	Either "y" (color by Y category, e.g. state – default) or the name of another column to map to fill (e.g. a residual column for diverging color).
<code>colors</code>	Optional character vector of fill colors. When <code>fill = "y"</code> , length must be at least the number of distinct y levels. Defaults to recycled Okabe-Ito.
<code>show_labels</code>	If TRUE, draw within-segment percentage labels.
<code>label_size</code>	Numeric size for segment labels.
<code>x_label, y_label</code>	Optional axis labels.

Details

Used internally by [plot_state_frequencies](#); exposed so that other plot methods (e.g. permutation-residual visualisations) can reuse the same geometry by supplying a different fill column.

Value

A ggplot object.

Examples

```
df <- data.frame(
  group = rep(c("A", "B", "C"), each = 3),
  state = rep(c("s1", "s2", "s3"), 3),
  count = c(10, 5, 2, 7, 8, 3, 4, 6, 12)
)
plot_mosaic(df, x = "group", y = "state", weight = "count")
```

`plot_state_frequencies`*Plot State Frequency Distributions*

Description

Visualise state (node) frequency distributions across groups for any Nestimate object that carries sequence data: a single netobject, a netobject_group, an mcml model, or an htna network.

Usage

```
plot_state_frequencies(x, ...)
```

```
## S3 method for class 'netobject'
```

```
plot_state_frequencies(  
  x,  
  style = "marimekko",  
  metric = "prop",  
  label = "prop",  
  legend = "auto",  
  legend_dir = "auto",  
  legend_frame = "none",  
  sort_states = "frequency",  
  colors = NULL,  
  label_size = 3.5,  
  abbreviate = FALSE,  
  include_macro = FALSE,  
  combine = "auto",  
  ncol = NULL,  
  node_groups = NULL,  
  ...  
)
```

```
## S3 method for class 'htna'
```

```
plot_state_frequencies(  
  x,  
  style = "marimekko",  
  metric = "prop",  
  label = "prop",  
  legend = "auto",  
  legend_dir = "auto",  
  legend_frame = "none",  
  sort_states = "frequency",  
  colors = NULL,  
  label_size = 3.5,  
  abbreviate = FALSE,  
  include_macro = FALSE,
```

```

        combine = "auto",
        ncol = NULL,
        node_groups = NULL,
        ...
    )

## S3 method for class 'mcml'
plot_state_frequencies(
    x,
    style = "marimekko",
    metric = "prop",
    label = "prop",
    legend = "auto",
    legend_dir = "auto",
    legend_frame = "none",
    sort_states = "frequency",
    colors = NULL,
    label_size = 3.5,
    abbreviate = FALSE,
    include_macro = FALSE,
    combine = "auto",
    ncol = NULL,
    node_groups = NULL,
    ...
)

## S3 method for class 'netobject_group'
plot_state_frequencies(
    x,
    style = "marimekko",
    metric = "prop",
    label = "prop",
    legend = "auto",
    legend_dir = "auto",
    legend_frame = "none",
    sort_states = "frequency",
    colors = NULL,
    label_size = 3.5,
    abbreviate = FALSE,
    include_macro = FALSE,
    combine = "auto",
    ncol = NULL,
    node_groups = NULL,
    ...
)

## Default S3 method:
plot_state_frequencies(x, ...)

```

Arguments

x	A netobject, netobject_group, mcml, or htna object.
...	Reserved for future use.
style	One of: • "marimekko" (default) – per-group treemap panels with cumulative-width geometry; tile area = within-group state share. • "bars" – horizontal bars sorted by frequency, faceted per group. For chi-square mosaics of a (group x state) contingency table, use mosaic_plot directly – it is kept as a separate function with its own dispatch surface.
metric	For style = "bars": which value the bar length encodes – "prop" (default) or "freq". Treemap and hierarchical-marimekko areas always encode proportion within group.
label	Inline tile / bar annotation. All formats render on a single line. • "prop" (default) – proportion only, e.g. "66%" • "freq" – count only, e.g. "1,234" • "both" – count + proportion, e.g. "1,234 (66%)" • "state" – state name only, e.g. "Average" • "all" – state + proportion, e.g. "Average (66%)" • "none" – no inline labels
legend	Legend position. "auto" (default) resolves per style: "none" for style = "bars" (the y-axis already names every state, so a colour legend is redundant); "per_facet" for htna/mcml treemaps (state vocabularies differ per panel, so each gets its own legend); "bottom" for single-network and netobject_group treemaps (shared state vocabulary, one shared legend). Override with any of "bottom", "top", "right", "left", "none", or "per_facet". The "per_facet" option requires the gridExtra package and returns a gtable.
legend_dir	Legend internal layout: "auto" (default – horizontal for top/bottom, vertical for left/right), or force "horizontal" or "vertical" regardless of position.
legend_frame	"none" (default) for an unframed legend, or "border" to draw a thin grey rectangle around the legend ("legend enclosed in a square").
sort_states	One of "frequency" (default – most frequent first), "alpha", or "none".
colors	Optional character vector overriding the default Okabe-Ito state palette. Length must be at least the number of unique states.
label_size	Numeric size of inline labels (max size when ggfittext is installed – text auto-shrinks per tile).
abbreviate	Abbreviate state names. FALSE (default) shows full names; TRUE truncates to the first 3 characters via <code>base::abbreviate()</code> (which extends the truncation as needed to keep names unique after collision); a positive integer sets the target minimum length explicitly (e.g. <code>abbreviate = 4</code>). Affects tile labels, legend, and the returned <code>\$table</code> .
include_macro	For mcml only: prepend a "macro" reference column showing aggregate state frequencies across all clusters. Default FALSE.

combine	For legend = "per_facet" only. "auto" (default) returns a single combined gtable for 1-3 panels and a list of ggplots (one per panel) for 4+ panels – many-cluster mcm1 layouts read better as separate figures than as a tile grid. TRUE forces a combined gtable via gridExtra ; FALSE forces a list (knitr renders each at the chunk's full fig.width / fig.height).
ncol	For legend = "per_facet" with combine = TRUE: number of columns in the grid arrangement. NULL (default) picks 1, 2, or 3 columns based on the number of panels.
node_groups	Optional named character vector mapping node labels to semantic groups. When supplied, panels (or bars) are coloured / annotated by group rather than by individual state, so state-level palettes can collapse onto a smaller categorical legend.

Details

The marimekko layout is dispatched per class:

- For mcm1, where states partition cleanly into clusters, the chart is a hierarchical 2D marimekko: cluster columns of width proportional to cluster total, segments stacked vertically with heights proportional to within-cluster state proportions.
- For all other classes (netobject, netobject_group, htna), each group is rendered as its own panel containing a squarified treemap: each state becomes a rectangular tile whose AREA is exactly proportional to the state's share within that group. Single-panel when no groups exist; faceted when groups are present.

The bar style produces horizontal bars (state on the y-axis), faceted by group when groups exist. All variants use the Okabe-Ito palette.

Value

A state_freq object: a list with the rendered \$plot (a ggplot or gtable), the tidy \$table (a data.frame with columns group, state, count, proportion), and the call's \$style, \$metric, \$source_class. The class supports print() (shows the tidy table in the console), plot() (renders the chart), and as.data.frame() (returns the table).

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  data(group_regulation_long, package = "Nestimate")
  nw <- build_network(group_regulation_long,
    method = "relative", format = "long",
    actor = "Actor", action = "Action",
    order = "Time", group = "Course")
  res <- plot_state_frequencies(nw)
  print(res)          # tidy frequency table in the console
  plot(res)           # ggplot chart
  head(as.data.frame(res))
}
```

predictability	<i>Compute Node Predictability</i>
----------------	------------------------------------

Description

Computes the proportion of variance explained (R^2) for each node in the network, following Haslbeck & Waldorp (2018).

For method = "glasso" or "pcor", predictability is computed analytically from the precision matrix:

$$R_j^2 = 1 - 1/\Omega_{jj}$$

where Ω is the precision (inverse correlation) matrix.

For method = "cor", predictability is the multiple R^2 from regressing each node on its network neighbors (nodes with non-zero edges).

Usage

```
predictability(object, ...)

## S3 method for class 'netobject'
predictability(object, data = NULL, ...)

## S3 method for class 'netobject_ml'
predictability(object, ...)

## S3 method for class 'netobject_group'
predictability(object, ...)
```

Arguments

object	A netobject or netobject_ml object.
...	Additional arguments (ignored).
data	Optional data frame of the original variables used to estimate the network. Required for method = "cor" (multiple- R^2 regression of each node on its neighbours); ignored for the precision-matrix path used by glasso/pcor, which has no need of the raw data.

Value

For netobject: a named numeric vector of R^2 values (one per node, between 0 and 1).

For netobject_ml: a list with elements \$between and \$within, each a named numeric vector.

A named numeric vector of predictability values per node.

A list with within and between predictability vectors.

A named list of per-group predictability vectors.

References

Haslbeck, J. M. B., & Waldorp, L. J. (2018). How well do network models predict observations? On the importance of predictability in network models. *Behavior Research Methods*, 50(2), 853–861. doi:10.3758/s134280170910x

Examples

```
set.seed(42)
mat <- matrix(rnorm(60), ncol = 4)
colnames(mat) <- LETTERS[1:4]
net <- build_network(as.data.frame(mat), method = "glasso")
predictability(net)
```

predict_links

Predict Missing or Future Links in a Network

Description

Computes link prediction scores for all node pairs using one or more structural similarity methods. Accepts netobject, mcml, cograph_network, or a raw weight matrix.

All methods are fully vectorized using matrix operations - no loops. Supports both weighted and binary adjacency, directed and undirected networks.

Usage

```
predict_links(
  x,
  methods = c("common_neighbors", "resource_allocation", "adamic_adar", "jaccard",
    "preferential_attachment", "katz"),
  weighted = TRUE,
  top_n = NULL,
  exclude_existing = TRUE,
  include_self = FALSE,
  katz_damping = NULL
)
```

Arguments

x	A netobject, mcml, cograph_network, or numeric square matrix.
methods	Character vector. One or more of: "common_neighbors", "resource_allocation", "adamic_adar", "jaccard", "preferential_attachment", "katz". Default: all six methods.
weighted	Logical. If TRUE, use the weight matrix directly instead of binarizing. Default: TRUE.

<code>top_n</code>	Integer or NULL. Return only the top N predictions per method. Default: NULL (all pairs).
<code>exclude_existing</code>	Logical. If TRUE, exclude node pairs that already have an edge. Default: TRUE.
<code>include_self</code>	Logical. If TRUE, include self-loop predictions. Default: FALSE.
<code>katz_damping</code>	Numeric or NULL. Attenuation factor for Katz index. If NULL, auto-computed as $0.9 / \text{spectral_radius}(A)$. Default: NULL.

Details

Methods:

common_neighbors Number of shared neighbors. For directed graphs, sums shared out-neighbors and shared in-neighbors. Vectorized as $A \% \% t(A) + t(A) \% \% A$.

resource_allocation Zhou et al. (2009). Like common neighbors but weights each shared neighbor z by $1/\text{degree}(z)$. Penalizes hubs, rewards rare shared connections.

adamic_adar Adamic & Adar (2003). Like resource allocation but weights by $1/\log(\text{degree}(z))$. Less aggressive penalty than RA.

jaccard Ratio of shared neighbors to total neighbors. For directed graphs, computed on combined (out+in) neighbor sets.

preferential_attachment Product of source out-degree and target in-degree. Captures the "rich-get-richer" effect.

katz Katz (1953). Weighted sum of all paths between nodes, exponentially damped by path length. Computed via matrix inversion: $(I - \text{beta} * A)^{-1} - I$. Captures global structure.

Value

An object of class "net_link_prediction" containing:

predictions Data frame with columns: from, to, method, score, rank. Sorted by score (descending) within each method.

scores Named list of score matrices (one per method).

methods Character vector of methods used.

nodes Character vector of node names.

directed Logical.

weighted Logical.

n_nodes Integer.

n_existing Integer. Number of existing edges.

References

Liben-Nowell, D. & Kleinberg, J. (2007). The link-prediction problem for social networks. *JASIST*, 58(7), 1019–1031.

Zhou, T., Lu, L. & Zhang, Y.-C. (2009). Network topology and link prediction. *European Physical Journal B*, 71, 623–630.

Adamic, L. A. & Adar, E. (2003). Friends and neighbors on the Web. *Social Networks*, 25(3), 211–230.

Katz, L. (1953). A new status index derived from sociometric analysis. *Psychometrika*, 18(1), 39–43.

See Also

[evaluate_links](#) for prediction evaluation, [build_network](#) for network estimation.

Examples

```
seqs <- data.frame(
  V1 = sample(LETTERS[1:5], 50, TRUE),
  V2 = sample(LETTERS[1:5], 50, TRUE),
  V3 = sample(LETTERS[1:5], 50, TRUE)
)
net <- build_network(seqs, method = "relative")
pred <- predict_links(net)
print(pred)
summary(pred)
```

```
prepare
```

```
Prepare Event Log Data for Network Estimation
```

Description

Converts event log data (actor, action, time) into wide sequence format suitable for [build_network](#). Automatically parses timestamps, detects sessions from time gaps, and handles tie-breaking.

Usage

```
prepare(
  data,
  actor,
  action,
  time = NULL,
  order = NULL,
  session = NULL,
  time_threshold = 900,
  custom_format = NULL,
  is_unix_time = FALSE,
  unix_time_unit = c("seconds", "milliseconds", "microseconds")
)
```

Arguments

<code>data</code>	Data frame with event log columns.
<code>actor</code>	Character or character vector. Column name(s) identifying who performed the action (e.g. "student" or <code>c("student", "group")</code>). If missing, all data is treated as one actor.
<code>action</code>	Character. Column name containing the action/state/code.
<code>time</code>	Character or NULL. Column name containing timestamps. Supports ISO8601, Unix timestamps (numeric), and 40+ date/time formats. If NULL, row order defines the sequence. Default: NULL.
<code>order</code>	Character or NULL. Column name for tie-breaking when timestamps are identical. If NULL, original row order is used. Default: NULL.
<code>session</code>	Character, character vector, or NULL. Column name(s) for explicit session grouping (e.g. "course" or <code>c("course", "semester")</code>). When combined with <code>time</code> , sessions are further split by time gaps. Default: NULL.
<code>time_threshold</code>	Numeric. Maximum gap in seconds between consecutive events before a new session starts. Only used when <code>time</code> is provided. Default: 900 (15 minutes).
<code>custom_format</code>	Character or NULL. Custom <code>strptime</code> format string for parsing timestamps. Default: NULL (auto-detect).
<code>is_unix_time</code>	Logical. If TRUE, treat numeric time values as Unix timestamps. Default: FALSE (auto-detected for numeric columns).
<code>unix_time_unit</code>	Character. Unit for Unix timestamps: "seconds", "milliseconds", or "microseconds". Default: "seconds".

Value

A list with class "nestimate_data" containing:

sequence_data Data frame in wide format (one row per session, columns T1, T2, ...).

long_data The processed long-format data with session IDs.

meta_data Session-level metadata (session ID, actor).

time_data Parsed time values in wide format (if time provided).

statistics List with `total_sessions`, `total_actions`, `max_sequence_length`, `unique_actors`, etc.

See Also

[build_network](#), [prepare_onehot](#)

Examples

```
df <- data.frame(
  student = rep(1:3, each = 5),
  code = sample(c("read", "write", "test"), 15, replace = TRUE),
  timestamp = seq.POSIXt(as.POSIXct("2024-01-01"), by = "min", length.out = 15)
)
prepared <- prepare(df, actor = "student", action = "code",
```

```

        time = "timestamp")
net <- build_network(prepared$sequence_data, method = "relative")

```

prepare_for_tna	<i>Prepare Data for TNA Analysis</i>
-----------------	--------------------------------------

Description

Prepare simulated or real data for use with `tna::tna()` and related functions. Handles various input formats and ensures the output is compatible with TNA models.

Usage

```

prepare_for_tna(
  data,
  type = c("sequences", "long", "auto"),
  state_names = NULL,
  id_col = "Actor",
  time_col = "Time",
  action_col = "Action",
  validate = TRUE
)

```

Arguments

<code>data</code>	Data frame containing sequence data.
<code>type</code>	Character. Type of input data: "sequences" Wide format with one row per sequence (default). "long" Long format with one row per action. "auto" Automatically detect format based on column names.
<code>state_names</code>	Character vector. Expected state names, or NULL to extract from data. Default: NULL.
<code>id_col</code>	Character. Name of ID column for long format data. Default: "Actor".
<code>time_col</code>	Character. Name of time column for long format data. Default: "Time".
<code>action_col</code>	Character. Name of action column for long format data. Default: "Action".
<code>validate</code>	Logical. Whether to validate that all actions are in <code>state_names</code> . Default: TRUE.

Details

This function performs several preparations:

1. Converts long format to wide format if needed.
2. Validates that all actions/states are recognized.
3. Removes any non-sequence columns (e.g., id, metadata).
4. Converts factors to characters.
5. Ensures consistent column naming (V1, V2, ...).

Value

A data frame ready for use with TNA functions. For "sequences" type, returns a data frame where each row is a sequence and columns are time points (V1, V2, ...). For "long" type, converts to wide format first.

See Also

[wide_to_long](#), [long_to_wide](#) for format conversions.

Examples

```
# From wide format sequences
sequences <- data.frame(
  V1 = c("A", "B", "C", "A"), V2 = c("B", "C", "A", "B"),
  V3 = c("C", "A", "B", "C"), V4 = c("A", "B", "A", "B")
)
tna_data <- prepare_for_tna(sequences, type = "sequences")
```

```
prepare_onehot
```

Import One-Hot Encoded Data into Sequence Format

Description

Converts binary indicator (one-hot) data into the wide sequence format expected by [build_network](#) and `tna::tna()`. Each binary column represents a state; rows where the value is 1 are marked with the column name. Supports optional windowed aggregation.

Simultaneous active states are preserved using the same window-span representation as `tna::import_onehot()`: each input row/window is expanded to one sequence slot per code and transition counting occurs between windows, not between simultaneous states inside the same row.

Usage

```
prepare_onehot(
  data,
  cols,
  actor = NULL,
  session = NULL,
  interval = NULL,
  window_size = 3L,
  window_type = c("non-overlapping", "overlapping"),
  aggregate = FALSE
)
```

Arguments

<code>data</code>	Data frame with binary (0/1) indicator columns.
<code>cols</code>	Character vector. Names of the one-hot columns to use.
<code>actor</code>	Character or NULL. Name of the actor/ID column. If NULL, all rows are treated as a single sequence. Default: NULL.
<code>session</code>	Character or NULL. Name of the session column for sub-grouping within actors. Default: NULL.
<code>interval</code>	Integer or NULL. Number of rows per time point in the output. If NULL, all rows become a single time point group. Default: NULL.
<code>window_size</code>	Integer (≥ 1). Number of consecutive rows to aggregate into each window. Default: 3. Set <code>window_size = 1</code> for no windowing (each row is its own time point).
<code>window_type</code>	Character. "non-overlapping" (fixed, separate windows) or "overlapping" (rolling, <code>step = 1</code>). Default: "non-overlapping".
<code>aggregate</code>	Logical. If TRUE, aggregate within each window by taking the first non-NA indicator per column. Default: FALSE.

Value

A data frame in wide format with columns named `W{window}_T{time}` where each cell contains a state name or NA. Attributes `windowed`, `window_size`, `window_span` are set on the result.

See Also

[action_to_onehot](#) for the reverse conversion.

Examples

```
# Simple binary data
df <- data.frame(
  A = c(1, 0, 1, 0, 1),
  B = c(0, 1, 0, 1, 0),
  C = c(0, 0, 0, 0, 0)
)
seq_data <- prepare_onehot(df, cols = c("A", "B", "C"))

# With actor grouping
df$actor <- c(1, 1, 1, 2, 2)
seq_data <- prepare_onehot(df, cols = c("A", "B", "C"), actor = "actor")

# With windowing
seq_data <- prepare_onehot(df, cols = c("A", "B", "C"),
                           window_size = 2, window_type = "non-overlapping")
```

print.boot_glasso	<i>Print Method for boot_glasso</i>
-------------------	-------------------------------------

Description

Print Method for boot_glasso

Usage

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

Arguments

x	A boot_glasso object.
...	Additional arguments (ignored).

Value

The input object, invisibly.

Examples

```
set.seed(1)  
dat <- as.data.frame(matrix(rnorm(60), ncol = 3))  
bg <- boot_glasso(dat, iter = 10, cs_iter = 5, centrality = "strength")  
print(bg)  
  
set.seed(42)  
mat <- matrix(rnorm(60), ncol = 4)  
colnames(mat) <- LETTERS[1:4]  
boot <- boot_glasso(as.data.frame(mat), iter = 20, cs_iter = 10,  
  centrality = "strength", seed = 42)  
print(boot)
```

print.chain_structure	<i>Print method for chain_structure</i>
-----------------------	---

Description

Prints a compact chain-level header. For the full per-state table, call `summary()` on the same object.

Usage

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

Arguments

x	A chain_structure object.
...	Ignored.

Value

x invisibly.

print.chain_structure_group
<i>Print method for chain_structure_group</i>

Description

One header line per group, followed by each group’s per-state table (via summary.chain_structure).

Usage

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

Arguments

x	A chain_structure_group (named list of chain_structure).
...	Forwarded to print.chain_structure.

Value

x invisibly.

print.cluster_choice	<i>Print Method for cluster_choice</i>
----------------------	--

Description

Print Method for cluster_choice

Usage

```
## S3 method for class 'cluster_choice'
print(x, digits = 3L, ...)
```

Arguments

x	A cluster_choice object.
digits	Integer. Decimal places for floating-point columns. Default 3L.
...	Unsupported. Supplying unused arguments raises an error.

Value

The input object, invisibly.

print.mcml	<i>Print Method for mcml</i>
------------	------------------------------

Description

Print Method for mcml

Usage

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

Arguments

x	An mcml object.
...	Unsupported. Supplying unused arguments raises an error.

Value

The input object, invisibly.

Examples

```
seqs <- data.frame(V1 = c("A", "B", "C", "A"), V2 = c("B", "C", "A", "B"))
clusters <- list(G1 = c("A", "B"), G2 = c("C"))
cs <- build_mcml(seqs, clusters)
print(cs)

seqs <- data.frame(
  T1 = c("A", "B", "A"), T2 = c("B", "C", "B"),
  T3 = c("C", "A", "C"), T4 = c("A", "B", "A")
)
clusters <- c("Alpha", "Beta", "Alpha")
cs <- build_mcml(seqs, clusters, type = "raw")
print(cs)
```

print.mcml_layer	<i>Print Method for an mcml Layer</i>
------------------	---------------------------------------

Description

Compact summary of one mcml layer (macro or a single within-cluster network) – nodes, edges, weight matrix dimensions – without spilling the full \$weights, \$inits, or \$data contents.

Usage

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

Arguments

x	An mcml_layer.
...	Unused.

Value

The input, invisibly.

print.mcml_pc	<i>Print an MCML-PC Object</i>
---------------	--------------------------------

Description

Print an MCML-PC Object

Usage

```
## S3 method for class 'mcml_pc'
print(x, digits = 3, ...)
```

Arguments

x	An mcml_pc object.
digits	Number of digits to display (default 3).
...	Additional arguments (ignored).

Value

x, invisibly.

print.mmm_compare	<i>Print Method for mmm_compare</i>
-------------------	-------------------------------------

Description

Print Method for mmm_compare

Usage

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

Arguments

x	An mmm_compare object.
...	Unsupported. Supplying unused arguments raises an error.

Value

The input object, invisibly.

Examples

```
seqs <- data.frame(V1 = sample(c("A","B","C"), 30, TRUE),
                   V2 = sample(c("A","B","C"), 30, TRUE))
cmp <- compare_mmm(seqs, k = 2:3, n_starts = 1, max_iter = 10, seed = 1)
print(cmp)

set.seed(1)
seqs <- data.frame(
  V1 = sample(c("A","B","C"), 30, TRUE),
  V2 = sample(c("A","B","C"), 30, TRUE),
  V3 = sample(c("A","B","C"), 30, TRUE)
)
cmp <- compare_mmm(seqs, k = 2:3, n_starts = 5, seed = 1)
print(cmp)
```

print.mosaic_analysis	<i>Print method for mosaic_analysis objects</i>
-----------------------	---

Description

Print method for mosaic_analysis objects

Usage

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

Arguments

x	A mosaic_analysis object.
...	Ignored.

Value

x, invisibly.

print.nestimate_data *Print Method for nestimate_data*

Description

Print Method for nestimate_data

Usage

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

Arguments

x	A nestimate_data object.
...	Additional arguments (ignored).

Value

The input object, invisibly.

Examples

```
events <- data.frame(  
  actor = c("u1", "u1", "u1", "u2", "u2", "u2"),  
  action = c("A", "B", "C", "B", "A", "C"),  
  time = c(1, 2, 3, 1, 2, 3)  
)  
nd <- prepare(events, action = "action",  
              actor = "actor", time = "time")  
print(nd)
```

print.netobject	<i>Print Method for Network Object</i>
-----------------	--

Description

Print Method for Network Object

Usage

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

Arguments

x	A netobject.
...	Additional arguments (ignored).

Value

The input object, invisibly.

Examples

```
seqs <- data.frame(V1 = c("A","B","C","A"), V2 = c("B","C","A","B"))
net <- build_network(seqs, method = "relative")
print(net)

seqs <- data.frame(
  V1 = c("A","B","A","C"), V2 = c("B","C","B","A"),
  V3 = c("C","A","C","B")
)
net <- build_network(seqs, method = "relative")
print(net)
```

print.netobject_group	<i>Print Method for Group Network Object</i>
-----------------------	--

Description

Compact summary of a netobject_group. Header surfaces the source (a clustering attached by [cluster_network](#) or [cluster_mmm](#), or a plain split by group_col). The per-group table carries node and edge counts, weight range, and – when a clustering attribute is present – N and percentage of sequences per cluster (matching the layout used by [print.net_clustering](#) and [print.net_mmm](#)).

Usage

```
## S3 method for class 'netobject_group'
print(x, digits = 3L, ...)
```

Arguments

x A `netobject_group`.

digits Integer. Decimal places for the weight summary. Default 3. Non-breaking: `print(x)` keeps the same shape as before, with the addition of a weight-range column.

... Additional arguments (ignored).

Value

The input object, invisibly.

Examples

```
seqs <- data.frame(V1 = c("A", "B", "A", "B"), V2 = c("B", "A", "B", "A"),
                  grp = c("X", "X", "Y", "Y"))
nets <- build_network(seqs, method = "relative", group = "grp")
print(nets)

seqs <- data.frame(
  V1 = c("A", "B", "A", "C", "B", "A"),
  V2 = c("B", "C", "B", "A", "C", "B"),
  V3 = c("C", "A", "C", "B", "A", "C"),
  grp = c("X", "X", "X", "Y", "Y", "Y")
)
nets <- build_network(seqs, method = "relative", group = "grp")
print(nets)
```

`print.netobject_ml` *Print Method for Multilevel Network Object*

Description

Print Method for Multilevel Network Object

Usage

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


Arguments

x A netobject_ml.
 ... Additional arguments (ignored).

Value

The input object, invisibly.

Examples

```
set.seed(1)
obs <- data.frame(id = rep(1:3, each = 5),
                  A = rnorm(15), B = rnorm(15), C = rnorm(15))
net_ml <- build_network(obs, method = "cor",
                       params = list(id = "id"), level = "both")
print(net_ml)

set.seed(1)
obs <- data.frame(
  id = rep(1:5, each = 8),
  A = rnorm(40), B = rnorm(40),
  C = rnorm(40), D = rnorm(40)
)
net_ml <- build_network(obs, method = "cor",
                       params = list(id = "id"), level = "both")
print(net_ml)
```

```
print.net_association_rules
```

Print Method for net_association_rules

Description

Print Method for net_association_rules

Usage

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

Arguments

x A net_association_rules object.
 ... Additional arguments (ignored).

Value

The input object, invisibly.

Examples

```
trans <- list(c("A","B","C"), c("A","B"), c("B","C","D"), c("A","C","D"))
rules <- association_rules(trans, min_support = 0.3, min_confidence = 0.5,
                           min_lift = 0)
print(rules)
```

print.net_bayes	<i>Print method for net_bayes</i>
-----------------	-----------------------------------

Description

Print method for net_bayes

Usage

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

Arguments

x	A net_bayes object.
...	Additional arguments (ignored).

Value

The input object, invisibly.

Examples

```
s1 <- data.frame(V1 = c("A","B","C"), V2 = c("B","C","A"))
s2 <- data.frame(V1 = c("A","C","B"), V2 = c("C","B","A"))
b <- bayes_compare(build_network(s1, method = "relative"),
                   build_network(s2, method = "relative"),
                   draws = 500, seed = 1)
print(b)
```

print.net_bayes_group *Print method for net_bayes_group*

Description

Print method for net_bayes_group

Usage

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

Arguments

x A net_bayes_group object.
... Additional arguments (ignored).

Value

The input object, invisibly.

Examples

```
s <- data.frame(V1 = c("A", "B", "A", "C"), V2 = c("B", "C", "B", "A"),  
               grp = c("X", "X", "Y", "Y"))  
nets <- build_network(s, method = "relative", group = "grp")  
print(bayes_compare(nets, draws = 200, seed = 1))
```

print.net_bootstrap *Print Method for net_bootstrap*

Description

Print Method for net_bootstrap

Usage

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

Arguments

x A net_bootstrap object.
... Additional arguments (ignored).

Value

The input object, invisibly.

Examples

```
net <- build_network(data.frame(V1 = c("A","B","C"), V2 = c("B","C","A")),
  method = "relative")
boot <- bootstrap_network(net, iter = 10)
print(boot)

set.seed(1)
seqs <- data.frame(
  V1 = c("A","B","A","C","B"), V2 = c("B","C","B","A","C"),
  V3 = c("C","A","C","B","A")
)
net <- build_network(seqs, method = "relative")
boot <- bootstrap_network(net, iter = 20)
print(boot)
```

```
print.net_bootstrap_group
```

Print Method for net_bootstrap_group

Description

Print Method for net_bootstrap_group

Usage

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

Arguments

x	A net_bootstrap_group object.
...	Ignored.

Value

x invisibly.

Examples

```
seqs <- data.frame(V1 = c("A", "B", "A", "C"), V2 = c("B", "C", "C", "A"),
  V3 = c("C", "A", "B", "B"), grp = c("X", "X", "Y", "Y"))
nets <- build_network(seqs, method = "relative", group = "grp")
boot <- bootstrap_network(nets, iter = 10)
print(boot)

set.seed(1)
seqs <- data.frame(
  V1 = c("A", "B", "A", "C", "B", "A"),
  V2 = c("B", "C", "B", "A", "C", "B"),
  V3 = c("C", "A", "C", "B", "A", "C"),
  grp = c("X", "X", "X", "Y", "Y", "Y")
)
nets <- build_network(seqs, method = "relative", group = "grp")
boot <- bootstrap_network(nets, iter = 20)
print(boot)
```

```
print.net_certainty    Print Method for net_certainty
```

Description

Print Method for net_certainty

Usage

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

Arguments

```
x                A net_certainty object.
...              Additional arguments (ignored).
```

Value

The input object, invisibly.

Examples

```
seqs <- data.frame(V1 = c("A", "B", "C"), V2 = c("B", "C", "A"))
print(certainty(build_network(seqs, method = "relative")))
```

print.net_clustering *Print Method for net_clustering*

Description

Compact, fixed-width summary of a sequence-clustering result. The header carries the clustering method and dissimilarity; the per-cluster table carries cluster size (count and percentage) and mean within-cluster distance when available. Optional medoid and covariate lines surface only when those fields are populated.

Usage

```
## S3 method for class 'net_clustering'
print(x, digits = 3L, ...)
```

Arguments

x	A net_clustering object.
digits	Integer. Decimal places used for floating-point statistics in the printout. Default 3. Non-breaking: existing print(x) calls keep their previous formatting.
...	Unsupported. Supplying unused arguments raises an error.

Value

The input object, invisibly.

Examples

```
seqs <- data.frame(V1 = c("A", "B", "C", "A", "B"), V2 = c("B", "C", "A", "B", "A"),
                  V3 = c("C", "A", "B", "C", "B"))
cl <- build_clusters(seqs, k = 2)
print(cl)

set.seed(1)
seqs <- data.frame(
  V1 = sample(c("A", "B", "C"), 20, TRUE),
  V2 = sample(c("A", "B", "C"), 20, TRUE),
  V3 = sample(c("A", "B", "C"), 20, TRUE)
)
cl <- build_clusters(seqs, k = 2)
print(cl)
```

```
print.net_cluster_diagnostics
```

Print Method for net_cluster_diagnostics

Description

Prints a uniform header, family-specific quality / IC line, and a per-cluster table. Layout matches [print.net_clustering](#) and [print.net_mmm](#).

Usage

```
## S3 method for class 'net_cluster_diagnostics'  
print(x, digits = 3L, ...)
```

Arguments

x	A net_cluster_diagnostics object.
digits	Integer. Decimal places for floating-point statistics. Default 3L.
...	Unsupported. Supplying unused arguments raises an error.

Value

The input object, invisibly.

```
print.net_gimme
```

Print Method for net_gimme

Description

Print Method for net_gimme

Usage

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

Arguments

x	A net_gimme object.
...	Additional arguments (ignored).

Value

The input object, invisibly.

Examples

```

set.seed(1)
panel <- data.frame(
  id = rep(1:5, each = 20),
  t = rep(seq_len(20), 5),
  A = rnorm(100), B = rnorm(100), C = rnorm(100)
)
gm <- build_gimme(panel, vars = c("A","B","C"), id = "id", time = "t")
print(gm)

```

print.net_hon

Print Method for net_hon

Description

Print Method for net_hon

Usage

```

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

```

Arguments

x A net_hon object.
 ... Additional arguments (ignored).

Value

The input object, invisibly.

Examples

```

seqs <- list(c("A","B","C","D"), c("A","B","C","A"), c("B","C","D","A"))
hon <- build_hon(seqs, max_order = 2)
print(hon)

```

```

seqs <- data.frame(
  V1 = c("A","B","C","A","B"),
  V2 = c("B","C","A","B","C"),
  V3 = c("C","A","B","C","A")
)
hon <- build_hon(seqs, max_order = 2L)
print(hon)

```

print.net_honem	<i>Print Method for net_honem</i>
-----------------	-----------------------------------

Description

Print Method for net_honem

Usage

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

Arguments

x	A net_honem object.
...	Additional arguments (ignored).

Value

The input object, invisibly.

Examples

```
seqs <- list(c("A","B","C","D"), c("A","B","C","A"), c("B","C","D","A"))
hem <- build_honem(build_hon(seqs, max_order = 2), dim = 2)
print(hem)

seqs <- data.frame(
  V1 = c("A","B","C","A","B"),
  V2 = c("B","C","A","B","C"),
  V3 = c("C","A","B","C","A")
)
hon <- build_hon(seqs, max_order = 2L)
honem <- build_honem(hon, dim = 2L)
print(honem)
```

print.net_hypa	<i>Print Method for net_hypa</i>
----------------	----------------------------------

Description

Print Method for net_hypa

Usage

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

Arguments

x A net_hypa object.
... Additional arguments (ignored).

Value

The input object, invisibly.

Examples

```
seqs <- list(c("A","B","C"), c("B","C","A"), c("A","C","B"), c("A","B","C"))
hyp <- build_hypa(seqs, k = 2)
print(hyp)
```

```
seqs <- data.frame(
  V1 = c("A","B","C","A","B","C","A","B","C","A"),
  V2 = c("B","C","A","B","C","A","B","C","A","B"),
  V3 = c("C","A","B","C","A","B","C","A","B","C"),
  V4 = c("A","B","C","A","B","C","A","B","C","A")
)
hypa <- build_hypa(seqs, k = 2L)
print(hypa)
```

```
print.net_link_prediction
```

Print Method for net_link_prediction

Description

Print Method for net_link_prediction

Usage

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

Arguments

x A net_link_prediction object.
... Additional arguments (ignored).

Value

The input object, invisibly.

Examples

```
seqs <- data.frame(
  V1 = sample(LETTERS[1:4], 30, TRUE),
  V2 = sample(LETTERS[1:4], 30, TRUE),
  V3 = sample(LETTERS[1:4], 30, TRUE)
)
net <- build_network(seqs, method = "relative")
pred <- predict_links(net)
print(pred)
```

```
print.net_markov_order
```

Print Method for net_markov_order

Description

Print Method for net_markov_order

Usage

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

Arguments

x	A net_markov_order object.
...	Ignored.

Value

The input object, invisibly.

Examples

```
# First-order Markov data: test should select order 1
set.seed(1)
states <- letters[1:4]
tm <- matrix(runif(16), 4, 4, dimnames = list(states, states))
tm <- tm / rowSums(tm)
seqs <- lapply(1:30, function(.) {
  s <- character(50); s[1] <- sample(states, 1)
  for (i in 2:50) s[i] <- sample(states, 1, prob = tm[s[i - 1], ])
  s
})
```

```

}))
res <- markov_order_test(seqs, max_order = 3, n_perm = 300, seed = 1)
res$optimal_order
summary(res)
plot(res)

```

```
print.net_markov_order_group
```

Print method for net_markov_order_group

Description

Print method for net_markov_order_group

Usage

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

Arguments

<code>x</code>	A net_markov_order_group (named list of net_markov_order results, one per group).
<code>...</code>	Forwarded to print.net_markov_order for each element.

Value

`x` invisibly.

```
print.net_markov_stability_group
```

Print method for net_markov_stability_group

Description

Print method for net_markov_stability_group

Usage

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

Arguments

- x A net_markov_stability_group (named list of net_markov_stability results).
- ... Forwarded to print.net_markov_stability for each element.

Value

x invisibly.

print.net_mlvar	<i>Print method for net_mlvar</i>
-----------------	-----------------------------------

Description

Print method for net_mlvar

Usage

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

Arguments

- x A net_mlvar object returned by [build_mlvar\(\)](#).
- ... Unused; present for S3 consistency.

Value

Invisibly returns x.

Examples

```
## Not run:
d <- simulate_data("mlvar", seed = 1)
fit <- build_mlvar(d, vars = attr(d, "vars"),
                  id = "id", day = "day", beep = "beep")

print(fit)
summary(fit)

## End(Not run)
```

print.net_mmm

Print Method for net_mmm

Description

Compact summary of a Mixed Markov Model fit. Header carries dimensions and information criteria; cluster table carries N, mixing share, and per-cluster average posterior probability (AvePP). Layout matches [print.net_clustering](#) so distance- and model-based clusterings can be compared at a glance.

Usage

```
## S3 method for class 'net_mmm'
print(x, digits = 3L, ...)
```

Arguments

x	A net_mmm object.
digits	Integer. Decimal places for floating-point statistics. Default 3. Non-breaking: print(x) keeps the same alignment as before.
...	Unsupported. Supplying unused arguments raises an error.

Value

The input object, invisibly.

Examples

```
seqs <- data.frame(V1 = sample(c("A", "B", "C"), 30, TRUE),
                  V2 = sample(c("A", "B", "C"), 30, TRUE))
mmm <- build_mmm(seqs, k = 2, n_starts = 1, max_iter = 10, seed = 1)
print(mmm)

set.seed(1)
seqs <- data.frame(
  V1 = sample(c("A", "B", "C"), 30, TRUE),
  V2 = sample(c("A", "B", "C"), 30, TRUE),
  V3 = sample(c("A", "B", "C"), 30, TRUE)
)
mmm <- build_mmm(seqs, k = 2, n_starts = 5, seed = 1)
print(mmm)
```

```
print.net_mmm_clustering
```

Print Method for MMM Clustering Attribute

Description

Prints the clustering metadata that `cluster_mmm` attaches to its `netobject_group` return value (`attr(grp, "clustering")`). Layout mirrors `print.net_clustering`: a one-line dimension header, a quality line with AvePP / entropy / classification error, information criteria, and a per-cluster table.

Usage

```
## S3 method for class 'net_mmm_clustering'
print(x, digits = 3L, ...)
```

Arguments

<code>x</code>	A <code>net_mmm_clustering</code> object.
<code>digits</code>	Integer. Decimal places for floating-point statistics. Default 3.
<code>...</code>	Unsupported. Supplying unused arguments raises an error.

Value

The input object, invisibly.

Examples

```
seqs <- data.frame(V1 = sample(c("A","B","C"), 30, TRUE),
                   V2 = sample(c("A","B","C"), 30, TRUE))
grp <- cluster_mmm(seqs, k = 2, n_starts = 1, max_iter = 10, seed = 1)
print(attr(grp, "clustering"))
```

```
print.net_mogen
```

Print Method for net_mogen

Description

Print Method for `net_mogen`

Usage

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

Arguments

x A net_mogen object.
 ... Additional arguments (ignored).

Value

The input object, invisibly.

Examples

```
seqs <- list(c("A","B","C","D"), c("A","B","C","A"), c("B","C","D","A"))
mg <- build_mogen(seqs, max_order = 2)
print(mg)
```

```
seqs <- data.frame(
  V1 = c("A","B","C","A","B"),
  V2 = c("B","C","A","B","C"),
  V3 = c("C","A","B","C","A")
)
mog <- build_mogen(seqs, max_order = 2L)
print(mog)
```

print.net_mpt_group *Print method for net_mpt_group*

Description

Print method for net_mpt_group

Usage

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

Arguments

x A net_mpt_group (named list of net_mpt results).
 ... Forwarded to print.net_mpt for each element.

Value

x invisibly.

print.net_nct	<i>Print Method for net_nct</i>
---------------	---------------------------------

Description

Print Method for net_nct

Usage

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

Arguments

x	A net_nct object.
...	Ignored.

Value

The input object, invisibly.

Examples

```
## Not run:
set.seed(1)
x1 <- matrix(rnorm(200 * 5), 200, 5)
x2 <- matrix(rnorm(200 * 5), 200, 5)
colnames(x1) <- colnames(x2) <- paste0("V", 1:5)
res <- nct(x1, x2, iter = 100)
res$M$p_value
res$S$p_value

## End(Not run)
```

print.net_path_dependence	<i>Print method for net_path_dependence</i>
---------------------------	---

Description

Print method for net_path_dependence

Usage

```
## S3 method for class 'net_path_dependence'
print(x, top = 10L, digits = 3L, ...)
```

Arguments

x	A net_path_dependence object.
top	Integer. Number of top contexts to show. Default 10.
digits	Integer. Digits to round numeric output. Default 3.
...	Ignored.

Value

x invisibly.

print.net_permutation *Print Method for net_permutation*

Description

Print Method for net_permutation

Usage

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

Arguments

x	A net_permutation object.
...	Additional arguments (ignored).

Value

The input object, invisibly.

Examples

```
s1 <- data.frame(V1 = c("A", "B", "C"), V2 = c("B", "C", "A"))
s2 <- data.frame(V1 = c("A", "C", "B"), V2 = c("C", "B", "A"))
n1 <- build_network(s1, method = "relative")
n2 <- build_network(s2, method = "relative")
perm <- permutation(n1, n2, iter = 10)
print(perm)

set.seed(1)
d1 <- data.frame(V1 = c("A", "B", "A"), V2 = c("B", "C", "B"),
                 V3 = c("C", "A", "C"))
d2 <- data.frame(V1 = c("C", "A", "C"), V2 = c("A", "B", "A"),
                 V3 = c("B", "C", "B"))
net1 <- build_network(d1, method = "relative")
net2 <- build_network(d2, method = "relative")
```

```
perm <- permutation(net1, net2, iter = 20, seed = 1)
print(perm)
```

```
print.net_permutation_group
```

Print Method for net_permutation_group

Description

Print Method for net_permutation_group

Usage

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

Arguments

x A net_permutation_group object.
... Additional arguments (ignored).

Value

x invisibly.

Examples

```
s1 <- data.frame(V1 = c("A", "B", "A", "C"), V2 = c("B", "C", "B", "A"),
  V3 = c("C", "A", "C", "B"), grp = c("X", "X", "Y", "Y"))
s2 <- data.frame(V1 = c("C", "A", "C", "B"), V2 = c("A", "B", "A", "C"),
  V3 = c("B", "C", "B", "A"), grp = c("X", "X", "Y", "Y"))
nets1 <- build_network(s1, method = "relative", group = "grp")
nets2 <- build_network(s2, method = "relative", group = "grp")
perm <- permutation(nets1, nets2, iter = 10)
print(perm)

set.seed(1)
s1 <- data.frame(V1 = c("A", "B", "A", "C"), V2 = c("B", "C", "B", "A"),
  V3 = c("C", "A", "C", "B"), grp = c("X", "X", "Y", "Y"))
s2 <- data.frame(V1 = c("C", "A", "C", "B"), V2 = c("A", "B", "A", "C"),
  V3 = c("B", "C", "B", "A"), grp = c("X", "X", "Y", "Y"))
nets1 <- build_network(s1, method = "relative", group = "grp")
nets2 <- build_network(s2, method = "relative", group = "grp")
perm <- permutation(nets1, nets2, iter = 20, seed = 1)
print(perm)
```

```
print.net_pruning_details
```

Print method for pruning details

Description

Print method for pruning details

Usage

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

Arguments

x	A net_pruning_details object.
...	Ignored.

Value

x, invisibly.

```
print.net_reliability
```

Print Method for net_reliability

Description

Print Method for net_reliability

Usage

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

Arguments

x	A net_reliability object.
...	Additional arguments (ignored).

Value

The input object, invisibly.

Examples

```
net <- build_network(data.frame(V1 = c("A","B","C","A"),
  V2 = c("B","C","A","B")), method = "relative")
rel <- network_reliability(net, iter = 10)
print(rel)

set.seed(1)
seqs <- data.frame(
  V1 = sample(c("A","B","C"), 30, TRUE),
  V2 = sample(c("A","B","C"), 30, TRUE),
  V3 = sample(c("A","B","C"), 30, TRUE)
)
net <- build_network(seqs, method = "relative")
rel <- network_reliability(net, iter = 20, seed = 1)
print(rel)
```

```
print.net_sequence_comparison
```

Print Method for net_sequence_comparison

Description

Print Method for net_sequence_comparison

Usage

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

Arguments

x	A net_sequence_comparison object.
...	Additional arguments (ignored).

Value

The input object, invisibly.

Examples

```
seqs <- data.frame(
  V1 = sample(LETTERS[1:4], 60, TRUE),
  V2 = sample(LETTERS[1:4], 60, TRUE),
  V3 = sample(LETTERS[1:4], 60, TRUE),
  V4 = sample(LETTERS[1:4], 60, TRUE)
)
grp <- rep(c("X", "Y"), 30)
```

```
net <- build_network(seqs, method = "relative")
res <- sequence_compare(net, group = grp, sub = 2:3, test = "chisq")
```

```
print.net_stability     Print Method for net_stability
```

Description

Print Method for net_stability

Usage

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

Arguments

x	A net_stability object.
...	Additional arguments (ignored).

Value

The input object, invisibly.

Examples

```
net <- build_network(data.frame(V1 = c("A","B","C","A"),
  V2 = c("B","C","A","B")), method = "relative")
cs <- centrality_stability(net, iter = 10, drop_prop = 0.3)
print(cs)

set.seed(1)
seqs <- data.frame(
  V1 = sample(c("A","B","C"), 30, TRUE),
  V2 = sample(c("A","B","C"), 30, TRUE),
  V3 = sample(c("A","B","C"), 30, TRUE)
)
net <- build_network(seqs, method = "relative")
stab <- centrality_stability(net, measures = c("InStrength","OutStrength"),
  iter = 10)

print(stab)
```

```
print.net_stability_group
```

Print Method for net_stability_group

Description

Print Method for net_stability_group

Usage

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

Arguments

x	A net_stability_group (returned by centrality_stability() when called on a netobject_group or an mcml).
...	Additional arguments (ignored).

Value

The input x invisibly.

```
print.net_transition_entropy
```

Print method for net_transition_entropy

Description

Print method for net_transition_entropy

Usage

```
## S3 method for class 'net_transition_entropy'
print(x, digits = 3, ...)
```

Arguments

x	A net_transition_entropy object.
digits	Integer. Digits to round numeric output. Default 3.
...	Ignored.

Value

x invisibly.

```
print.net_transition_entropy_group
```

Print method for net_transition_entropy_group

Description

Print method for net_transition_entropy_group

Usage

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

Arguments

x	A net_transition_entropy_group.
...	Forwarded to print.net_transition_entropy.

Value

x invisibly.

```
print.net_vertex_bootstrap
```

Print a Vertex Bootstrap Result

Description

Print a Vertex Bootstrap Result

Usage

```
## S3 method for class 'net_vertex_bootstrap'  
print(x, digits = 3, ...)
```

Arguments

x	A net_vertex_bootstrap object.
digits	Number of digits to display (default 3).
...	Additional arguments (ignored).

Value

x, invisibly.

```
print.net_vertex_comparison
```

Print a Two-Network Vertex-Bootstrap Comparison

Description

Print a Two-Network Vertex-Bootstrap Comparison

Usage

```
## S3 method for class 'net_vertex_comparison'  
print(x, digits = 3, ...)
```

Arguments

x	A net_vertex_comparison object.
digits	Number of digits to display (default 3).
...	Additional arguments (ignored).

Value

x, invisibly.

```
print.pc_loading_stability
```

Print Composite-Weight Stability

Description

Print Composite-Weight Stability

Usage

```
## S3 method for class 'pc_loading_stability'  
print(x, digits = 3, ...)
```

Arguments

x	A pc_loading_stability object.
digits	Number of digits to display (default 3).
...	Additional arguments (ignored).

Value

x, invisibly.

```
print.persistence_landscape
```

Print Persistence Landscape

Description

Print Persistence Landscape

Usage

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

Arguments

x	A persistence_landscape object.
...	Ignored.

Value

The input, invisibly.

Examples

```
mat <- matrix(c(0, .6, .5, .6, 0, .4, .5, .4, 0), 3, 3)
rownames(mat) <- colnames(mat) <- c("A", "B", "C")
ph <- persistent_homology(mat, n_steps = 5)
pl <- persistence_landscape(ph, k_max = 3, dimension = 0)
print(pl)
```

```
print.persistent_homology
```

Print persistent homology results

Description

Print persistent homology results

Usage

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

Arguments

x	A persistent_homology object.
...	Additional arguments (unused).

Value

The input object, invisibly.

Examples

```
mat <- matrix(c(0,.6,.5,.6,0,.4,.5,.4,0), 3, 3)
colnames(mat) <- rownames(mat) <- c("A","B","C")
ph <- persistent_homology(mat, n_steps = 10)
print(ph)
```

print.q_analysis	<i>Print Q-analysis results</i>
------------------	---------------------------------

Description

Print Q-analysis results

Usage

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

Arguments

x	A q_analysis object.
...	Additional arguments (unused).

Value

The input object, invisibly.

Examples

```
mat <- matrix(c(0,.6,.5,.6,0,.4,.5,.4,0), 3, 3)
colnames(mat) <- rownames(mat) <- c("A","B","C")
sc <- build_simplicial(mat, threshold = 0.3)
qa <- q_analysis(sc)
print(qa)
```

```
print.simplicial_complex
```

Print a simplicial complex

Description

Print a simplicial complex

Usage

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

Arguments

x	A simplicial_complex object.
...	Additional arguments (unused).

Value

The input object, invisibly.

Examples

```
mat <- matrix(c(0,.6,.5,.6,0,.4,.5,.4,0), 3, 3)
colnames(mat) <- rownames(mat) <- c("A","B","C")
sc <- build_simplicial(mat, threshold = 0.3)
print(sc)
```

```
print.summary.net_path_dependence
```

Print method for summary.net_path_dependence

Description

Print method for summary.net_path_dependence

Usage

```
## S3 method for class 'summary.net_path_dependence'
print(x, digits = 3L, ...)
```

Arguments

x	A summary.net_path_dependence object.
digits	Integer. Digits to round numeric output. Default 3.
...	Ignored.

Value

x invisibly.

```
print.summary.net_transition_entropy
```

Print method for summary.net_transition_entropy

Description

Print method for summary.net_transition_entropy

Usage

```
## S3 method for class 'summary.net_transition_entropy'
print(x, digits = 3, ...)
```

Arguments

x	A summary.net_transition_entropy.
digits	Integer. Digits to round numeric output. Default 3.
...	Ignored.

Value

x invisibly.

```
print.summary_chain_structure
```

Print method for summary.chain_structure

Description

Prints a one-line chain header followed by the tidy per-state table.

Usage

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

Arguments

<code>x</code>	A <code>summary_chain_structure</code> object.
<code>...</code>	Forwarded to <code>print.data.frame</code> .

Value

`x` invisibly.

`print.tidy_covariates` *Print method for tidy covariate output*

Description

Prints a one-line header naming the estimator, then the `data.frame`. The full human-readable view (per-cluster stats, profiles, OR/test tables) was already printed by `summary()` when this object was produced, so this method intentionally stays minimal to avoid duplication. Auto-prints when the user types the variable at the REPL.

Usage

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

Arguments

<code>x</code>	A <code>tidy_covariates</code> <code>data.frame</code> .
<code>...</code>	Ignored.

Value

The input invisibly.

`print.wtna_boot_mixed` *Print Method for wtna_boot_mixed*

Description

Print Method for `wtna_boot_mixed`

Usage

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

Arguments

x A wtna_boot_mixed object.
 ... Additional arguments (ignored).

Value

The input object, invisibly.

Examples

```
oh <- data.frame(A = c(1,0,1,0), B = c(0,1,0,1), C = c(1,1,0,0))
mixed <- wtna(oh, method = "both")
boot <- bootstrap_network(mixed, iter = 10)
print(boot)

set.seed(1)
oh <- data.frame(
  A = c(1,0,1,0,1,0,1,0),
  B = c(0,1,0,1,0,1,0,1),
  C = c(1,1,0,0,1,1,0,0)
)
mixed <- wtna(oh, method = "both")
boot <- bootstrap_network(mixed, iter = 20)
print(boot)
```

print.wtna_mixed	<i>Print Method for wtna_mixed</i>
------------------	------------------------------------

Description

Print Method for wtna_mixed

Usage

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

Arguments

x A wtna_mixed object.
 ... Additional arguments (ignored).

Value

The input object, invisibly.

Examples

```
oh <- matrix(c(1,0,0, 0,1,0, 0,0,1, 1,0,0), nrow = 4, byrow = TRUE,
             dimnames = list(NULL, c("A","B","C")))
mixed <- wtna(oh, method = "both")
print(mixed)
```

```
oh <- data.frame(
  A = c(1,0,1,0,1,0,1,0),
  B = c(0,1,0,1,0,1,0,1),
  C = c(1,1,0,0,1,1,0,0)
)
mixed <- wtna(oh, method = "both")
print(mixed)
```

```
print.wtna_perm_mixed Print Method for wtna_perm_mixed
```

Description

Print Method for wtna_perm_mixed

Usage

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

Arguments

<code>x</code>	A wtna_perm_mixed object.
<code>...</code>	Additional arguments (ignored).

Value

The input object, invisibly.

q_analysis	<i>Q-Analysis</i>
------------	-------------------

Description

Computes Q-connectivity structure (Atkin 1974). Two maximal simplices are q-connected if they share a face of dimension $\geq q$. Reports:

- **Q-vector:** number of connected components at each q-level
- **Structure vector:** highest simplex dimension per node

Usage

```
q_analysis(sc)
```

Arguments

sc A simplicial_complex object.

Value

A q_analysis object with \$q_vector, \$structure_vector, and \$max_q.

References

Atkin, R. H. (1974). *Mathematical Structure in Human Affairs*.

Examples

```
mat <- matrix(c(0,.6,.5,.6,0,.4,.5,.4,0), 3, 3)
colnames(mat) <- c("A","B","C")
sc <- build_simplicial(mat, threshold = 0.3)
q_analysis(sc)
```

register_estimator	<i>Register a Network Estimator</i>
--------------------	-------------------------------------

Description

Register a custom or built-in network estimator function by name. Estimators registered here can be used by [estimate_network](#) via the method parameter.

Usage

```
register_estimator(name, fn, description, directed)
```

Arguments

name	Character. Unique name for the estimator (e.g. "relative", "glasso").
fn	Function. The estimator function. Must accept data as its first argument and ... for additional parameters. Must return a list with at least: matrix (square numeric matrix), nodes (character vector), directed (logical).
description	Character. Short description of the estimator.
directed	Logical. Whether the estimator produces directed networks.

Value

Invisible NULL.

See Also

[get_estimator](#), [list_estimators](#), [remove_estimator](#), [estimate_network](#)

Examples

```
my_fn <- function(data, ...) {
  m <- cor(data)
  diag(m) <- 0
  list(matrix = m, nodes = colnames(m), directed = FALSE)
}
register_estimator("my_cor", my_fn, "Custom correlation", directed = FALSE)
df <- data.frame(A = rnorm(20), B = rnorm(20), C = rnorm(20))
net <- build_network(df, method = "my_cor")
remove_estimator("my_cor")
```

remove_estimator	<i>Remove a Registered Estimator</i>
------------------	--------------------------------------

Description

Remove a network estimator from the registry.

Usage

```
remove_estimator(name)
```

Arguments

name	Character. Name of the estimator to remove.
------	---

Value

Invisible NULL.

See Also

[register_estimator](#), [list_estimators](#)

Examples

```
register_estimator("test_est", function(data, ...) diag(3),
  description = "test", directed = FALSE)
remove_estimator("test_est")
```

rename_models	<i>Rename the models of a netobject_group</i>
---------------	---

Description

Replaces the names of the constituent networks in a `netobject_group` (or any object inheriting from it). Useful when `build_network()` produced generic labels (e.g. "Cluster 1", "Cluster 2") and you want to substitute meaningful ones (e.g. "High engagement", "Low engagement").

Usage

```
rename_models(x, new_names)

## S3 method for class 'netobject_group'
rename_models(x, new_names)

## Default S3 method:
rename_models(x, new_names)
```

Arguments

<code>x</code>	A <code>netobject_group</code> (or any object inheriting from it, such as <code>net_mlvar</code>).
<code>new_names</code>	A character vector of new names. Must have the same length as <code>x</code> , contain no NA or empty strings, and be unique.

Value

A `netobject_group` of the same class with renamed members.

Examples

```
## Not run:
d <- tna::group_regulation
grp <- build_network(d, method = "tna",
  group = sample(c("a", "b"), nrow(d), TRUE))
grp <- rename_models(grp, c("High", "Low"))
names(grp)

## End(Not run)
```

sequence_compare

*Compare Subsequence Patterns Between Groups***Description**

Extracts all k-gram patterns (subsequences of length k) from sequences in each group, computes standardized residuals against the independence model, and optionally runs a permutation or chi-square test of group differences.

Usage

```
sequence_compare(
  x,
  group = NULL,
  sub = 3:5,
  min_freq = 5L,
  test = c("permutation", "chisq", "none"),
  iter = 1000L,
  adjust = "fdr"
)
```

Arguments

x	A netobject_group (from grouped build_network), a netobject (requires group), or a wide-format data.frame (requires group).
group	Character or vector. Column name or vector of group labels. Not needed for netobject_group.
sub	Integer vector. Pattern lengths to analyze. Default: 3:5.
min_freq	Integer. Minimum frequency in each group for a pattern to be included. Default: 5.
test	Character. Inference method: one of "permutation" (default), "chisq", or "none". See Details.
iter	Integer. Permutation iterations. Only used when test = "permutation". Default: 1000.
adjust	Character. P-value correction method (see p.adjust). Default: "fdr".

Details

Standardized residuals are always computed from a 2xG contingency table of (this pattern vs. everything else) using the textbook formula $(o - e) / \sqrt{e * (1 - r/N) * (1 - c/N)}$. They describe how much each group's count for a given pattern deviates from expectation under independence, scaled to be approximately $N(0,1)$ under the null.

The optional test argument chooses an inference method:

"permutation" Shuffles group labels across sequences and recomputes a per-pattern statistic (row-wise Euclidean residual norm). Answers: "is this pattern's distribution associated with group membership at the *actor* level?" Respects the sequence as the unit of analysis; can be under-powered when the number of sequences is small.

"chisq" Runs `chisq.test` on the 2xG table per pattern. Answers: "do the group *streams* generate this pattern at different rates?" Treats each k-gram occurrence as an event; fast and powerful even with few sequences, but the iid assumption it makes is optimistic when sequences are strongly autocorrelated.

"none" Skip inference. Only residuals, frequencies, and proportions are returned.

P-values are adjusted once across all patterns (not per-pattern) using any method supported by `p.adjust`. The default is "fdr" (Benjamini-Hochberg).

Value

An object of class "net_sequence_comparison" containing:

patterns Tidy data.frame. Always present: `pattern`, `length`, `freq_<group>`, `prop_<group>`, `resid_<group>`. If `test = "permutation"`: `effect_size`, `p_value`. If `test = "chisq"`: `statistic`, `p_value`.

groups Character vector of group names.

n_patterns Integer. Number of patterns passing `min_freq`.

params List of `sub`, `min_freq`, `test`, `iter`, `adjust`.

Examples

```
seqs <- data.frame(
  V1 = sample(LETTERS[1:4], 60, TRUE),
  V2 = sample(LETTERS[1:4], 60, TRUE),
  V3 = sample(LETTERS[1:4], 60, TRUE),
  V4 = sample(LETTERS[1:4], 60, TRUE)
)
grp <- rep(c("X", "Y"), 30)
net <- build_network(seqs, method = "relative")
res <- sequence_compare(net, group = grp, sub = 2:3, test = "chisq")
```

sequence_plot

Sequence Plot (heatmap, index, or distribution)

Description

Single entry point for three categorical-sequence visualisations.

- `type = "heatmap"` (default): dense carpet, rows reordered by `sort` / dendrogram (single panel).
- `type = "index"`: same data layout, but rows separated by thin gaps (no dendrogram). Supports grouping via `group` or a `net_clustering`, plus a `ncol` x `nrow` facet grid.
- `type = "distribution"`: dispatches to [distribution_plot](#).

Usage

```
sequence_plot(
  x,
  type = c("heatmap", "index", "distribution"),
  sort = c("lcs", "frequency", "start", "end", "hamming", "osa", "lv", "dl", "qgram",
    "cosine", "jaccard", "jw"),
  tree = NULL,
  group = NULL,
  scale = c("proportion", "count"),
  geom = c("area", "bar"),
  na = TRUE,
  normalize = FALSE,
  trim = NULL,
  trim_clusterwise = FALSE,
  row_gap = 0,
  dendrogram_width = 1.2,
  k = NULL,
  k_color = "white",
  k_line_width = 2.5,
  state_colors = NULL,
  na_color = "grey90",
  cell_border = NA,
  frame = FALSE,
  width = NULL,
  height = NULL,
  main = NULL,
  show_n = TRUE,
  time_label = "Time",
  xlab = NULL,
  y_label = NULL,
  ylab = NULL,
  tick = NULL,
  ncol = NULL,
  nrow = NULL,
  combined = TRUE,
  legend = NULL,
  legend_size = NULL,
  legend_title = NULL,
  legend_ncol = NULL,
  legend_border = NA,
  legend_bty = "n"
)
```

Arguments

x Wide-format sequence data. Accepts:
data.frame / matrix Rows = sequences, columns = time points.
netobject Extracts \$data.

net_clustering	From <code>build_clusters</code> . Uses <code>\$data</code> , <code>\$assignments</code> for grouping, and <code>\$distance</code> for dendrogram.
netobject_group	From <code>cluster_network</code> or <code>build_network</code> on a clustering. Extracts data and assignments from <code>attr(, "clustering")</code> .
net_mmm	From <code>build_mmm</code> . Uses <code>\$models[[1]]\$data</code> and <code>\$assignments</code> .
tna	From the <code>tna</code> package. Decodes integer-encoded sequences.
mcml	From <code>build_mcml</code> (built from sequences). Produces a multichannel plot: one panel per cluster plus a macro Summary panel. <code>type = "heatmap"/"index"</code> draw the carpet (each channel's own states solid, other clusters a faded wash); <code>type = "distribution"</code> draws the stacked distribution (add <code>normalize = TRUE</code> for a TraMineR-style <code>seqdplot</code> where each time point sums to 1). Returns a <code>ggplot</code> object.
type	One of "heatmap" (default), "index", or "distribution".
sort	Row-ordering strategy for heatmap / within-panel for index. One of "lcs" (default), "frequency", "start", "end", or any <code>build_clusters</code> distance ("hamming", "osa", "lv", "dl", "qgram", "cosine", "jaccard", "jw").
tree	Optional <code>hclust</code> /dendrogram/ <code>agnes</code> object to supply row ordering (heatmap only; overrides sort).
group	Optional grouping vector (length <code>nrow(x)</code>) producing one facet per group. Index/distribution only. Ignored for heatmap.
scale, geom, na	Passed to <code>distribution_plot</code> when <code>type = "distribution"</code> .
normalize	<code>mcml + type = "distribution"</code> only. When <code>TRUE</code> , each time point is normalised to sum to 1 within its channel (TraMineR-style <code>seqdplot</code> composition); when <code>FALSE</code> (default) the stack shows prevalence and is capped with an NA band.
trim	Optional time-axis truncation, to stop a few long sequences from stretching the plot. Applies to all three types (including the <code>mcml</code> multichannel view). <code>NULL</code> (default) plots the full width. A fraction in $(0, 1)$ drops everything past that quantile of sequence lengths (e.g. <code>trim = 0.95</code> keeps the columns covering the shortest 95\% ≥ 1 is an absolute cut (<code>trim = 50</code> keeps the first 50 time points).
trim_clusterwise	Grouped <code>type = "index" / "distribution"</code> only, and only when <code>trim</code> is a fraction. <code>FALSE</code> (default) computes one cutoff on the pooled data and applies it to every panel, so all facets share the same width and the time axes stay aligned. <code>TRUE</code> crops each group to its own length quantile, so panels can end up at different widths (ragged axes). Absolute <code>trim (≥ 1)</code> ignores this - the column is the same everywhere either way.
row_gap	Fraction of row height used as vertical gap between sequences in index plots. 0 (default) = dense like heatmap. Try 0.15 for visible separators at low row counts.
dendrogram_width	Width ratio of the dendrogram panel (heatmap).
k	Optional integer. When supplied in <code>type = "heatmap"</code> , cuts the dendrogram into <code>k</code> clusters and draws thin horizontal separators between them in the carpet. Ignored when there is no dendrogram (e.g. <code>sort = "start"</code>) or for other types.
k_color	Colour for the cluster separator lines. Default "white".

k_line_width	Line width for the cluster separators. Default 2.5.
state_colors	Vector of colours, one per state.
na_color	Colour for NA cells.
cell_border	Cell border colour. NA = off.
frame	If TRUE (default), draw a box around each panel. If FALSE, no box - axis ticks and labels still appear.
width, height	Optional device dimensions in inches. When supplied, opens a new graphics device via <code>grDevices::dev.new()</code> . In knitr chunks use the <code>fig.width / fig.height</code> chunk options instead.
main	Plot title.
show_n	Append "(n = N)" to the title.
time_label, xlab	X-axis label. xlab is an alias.
y_label, ylab	Y-axis label (distribution only). ylab alias.
tick	Show every Nth x-axis label. NULL = auto.
ncol, nrow	Facet grid dimensions (index + distribution). Ignored when combined = FALSE.
combined	Index and distribution types only. When TRUE (default), groups are arranged on one figure via <code>graphics::layout()</code> . When FALSE, each group is drawn on its own page (one full-size figure per group, with its own legend). Single-group calls (<code>G == 1</code>) ignore this argument. Heatmap is always single-figure.
legend	Legend position: "bottom", "right", or "none". Default varies by type.
legend_size	Legend text size. NULL (default) auto-scales from the device width so the legend looks proportional at 5 in vs 12 in figures (clamped to <code>[0.65, 1.2]</code>).
legend_title	Optional legend title.
legend_ncol	Number of legend columns.
legend_border	Swatch border colour.
legend_bty	"n" or "o".

Value

For base-graphics types, invisibly a list describing the plot (shape depends on type). For an `mcml` input, a `ggplot` object.

See Also

[distribution_plot](#), [build_clusters](#), [build_mcml](#)

Examples

```
sequence_plot(trjectories)
sequence_plot(trjectories, type = "index")
sequence_plot(trjectories, type = "distribution")

# Multichannel MCML view: one channel per cluster + a macro Summary.
```



```

fit <- build_mcml(
  group_regulation_long,
  clusters = list(Cognitive = c("discuss", "synthesis", "consensus", "cohesion"),
    Regulation = c("plan", "monitor", "adapt", "coregulate"),
    Affective = "emotion"),
  actor = "Actor", action = "Action", time = "Time")
sequence_plot(fit) # multichannel carpet
sequence_plot(fit, type = "distribution") # prevalence + NA band
sequence_plot(fit, type = "distribution", normalize = TRUE) # seqdplot (sums to 1)

```

simplicial_degree	<i>Simplicial Degree</i>
-------------------	--------------------------

Description

Counts how many simplices of each dimension contain each node.

Usage

```
simplicial_degree(sc, normalized = FALSE)
```

Arguments

sc	A simplicial_complex object.
normalized	Divide by maximum possible count. Default FALSE.

Value

Data frame with node, columns d0 through d_k, and total (sum of d1+). Sorted by total descending.

Examples

```

mat <- matrix(c(0,.6,.5,.6,0,.4,.5,.4,0), 3, 3)
colnames(mat) <- rownames(mat) <- c("A","B","C")
sc <- build_simplicial(mat, threshold = 0.3)
simplicial_degree(sc)

```

srl_strategies	<i>Self-Regulated Learning Strategy Frequencies</i>
----------------	---

Description

Simulated frequency counts of 9 self-regulated learning (SRL) strategies for 250 university students. Strategies are grouped into three clusters: metacognitive (Planning, Monitoring, Evaluating), cognitive (Elaboration, Organization, Rehearsal), and resource management (Help_Seeking, Time_Mgmt, Effort_Reg). Within-cluster correlations are moderate (0.3–0.6), cross-cluster correlations are weaker.

Usage

```
srl_strategies
```

Format

A data frame with 250 rows and 9 columns. Each column is an integer count of how often the student used that strategy.

Examples

```
net <- build_network(srl_strategies, method = "glasso",
                    params = list(gamma = 0.5))
net
```

state_distribution	<i>Per-Class State Distribution as a Tidy Data Frame</i>
--------------------	--

Description

Returns a tidy data.frame(group, state, count, proportion) with one row per (group, state) cell. Companion to [state_frequencies](#) (which counts unique states in raw sequence input); state_distribution() pulls the same shape of frame from a fitted Nestimate object so analyses don't have to reach for the underlying \$data slot directly.

Usage

```
state_distribution(x, ...)

## S3 method for class 'netobject'
state_distribution(x, ...)

## S3 method for class 'htna'
state_distribution(x, ...)
```

```
## S3 method for class 'mcml'
state_distribution(x, include_macro = FALSE, ...)

## S3 method for class 'netobject_group'
state_distribution(x, ...)

## Default S3 method:
state_distribution(x, ...)
```

Arguments

x	A netobject, netobject_group, mcml, or htna object.
...	Currently unused.
include_macro	For mcml: when TRUE, prepend a group = "macro" block aggregating across clusters. Ignored for the other classes.

Details

Used internally by [plot_state_frequencies](#) as the data layer behind every chart, and surfaced as the \$table slot of the returned state_freq object.

Value

A data.frame with columns group (character), state (character), count (integer), and proportion (numeric, within-group share).

Examples

```
## Not run:
data(ai_long)
net <- build_network(ai_long, method = "frequency",
                    id_col = "session_id",
                    time_col = "order_in_session", action = "code")
state_distribution(net)

## End(Not run)
```

state_freq

Print, Plot, and Convert a state_freq Object

Description

plot_state_frequencies() returns a state_freq object holding both the rendered chart and the tidy frequency table. print() shows the table in the console, plot() renders the chart, and as.data.frame() returns the tidy table for downstream piping.

Usage

```
## S3 method for class 'state_freq'
print(x, digits = 1, max_states = 20L, ...)

## S3 method for class 'state_freq'
plot(x, ...)

## S3 method for class 'state_freq'
as.data.frame(x, ...)
```

Arguments

x	A state_freq object.
digits	Number of decimal places for proportion / share columns.
max_states	Cap on rows shown per group in the per-state table. The full table remains available via x\$table.
...	Unused.

Value

print() returns invisible(x); plot() returns invisible(NULL) after drawing; as.data.frame() returns x\$table.

state_frequencies	<i>Compute State Frequencies from Trajectory Data</i>
-------------------	---

Description

Counts how often each state appears across all trajectories. Returns a data frame sorted by frequency (descending).

Usage

```
state_frequencies(data)
```

Arguments

data	A list of character vectors (trajectories) or a data.frame.
------	---

Value

A data frame with columns: state, count, proportion.

Examples

```
trajs <- list(c("A","B","C"), c("A","B","A"))
state_frequencies(trajs)
```

subtract_networks	<i>Subtract one network from another</i>
-------------------	--

Description

Returns $x - y$ as a `netdifference` object: the element-wise difference of the two weight matrices. Works on any pair of networks; for an edge-betweenness difference, subtract two [net_edge_betweenness](#) results. Draw the signed difference network with `cograph::splot(d)` or `cograph::plot_difference(d)`; `cograph` handles the colouring and node palette.

Usage

```
subtract_networks(x, y)

## S3 method for class 'netdifference'
print(x, max_print = 12L, ...)
```

Arguments

<code>x, y</code>	A <code>netobject</code> , <code>cograph_network</code> , or numeric square matrix. Both must share the same nodes in the same order.
<code>max_print</code>	Integer. Rows to show in <code>print()</code> . Default 12.
<code>...</code>	Ignored.

Value

A `netdifference` object: a `netobject` whose `$weights` and `$difference_matrix` are $x - y$, carrying the source matrices `$x` and `$y`.

Examples

```
seqs <- data.frame(
  V1 = c("A", "B", "A", "C", "B", "A"), V2 = c("B", "C", "B", "A", "C", "B"),
  V3 = c("C", "A", "C", "B", "A", "C"))
a <- build_network(seqs, method = "relative")
b <- build_network(seqs[1:4, ], method = "relative")
subtract_networks(a, b)
# edge-betweenness difference:
subtract_networks(net_edge_betweenness(a), net_edge_betweenness(b))
```

summary.boot_glasso	<i>Summary Method for boot_glasso</i>
---------------------	---------------------------------------

Description

Summary Method for boot_glasso

Usage

```
## S3 method for class 'boot_glasso'  
summary(object, type = "edges", ...)
```

Arguments

object	A boot_glasso object.
type	Character. Summary type: "edges" (default), "centrality", "cs", "predictability", or "all".
...	Additional arguments (ignored).

Value

A data frame or list of data frames depending on type.

Examples

```
set.seed(1)  
dat <- as.data.frame(matrix(rnorm(60), ncol = 3))  
bg <- boot_glasso(dat, iter = 10, cs_iter = 5, centrality = "strength")  
summary(bg, type = "edges")  
  
set.seed(42)  
mat <- matrix(rnorm(60), ncol = 4)  
colnames(mat) <- LETTERS[1:4]  
boot <- boot_glasso(as.data.frame(mat), iter = 20, cs_iter = 10,  
  centrality = "strength", seed = 42)  
summary(boot, type = "edges")
```

summary.chain_structure

Tidy per-state summary of a chain_structure

Description

Returns a single data.frame with one row per state, combining every per-state metric chain_structure() computes. Always includes state, classification, period, return_probability (the diagonal of the hitting matrix) and persistence (the diagonal of the transition matrix); adds sojourn whenever it is finite, the chain's stationary_probability when irreducible, and absorption columns when the chain has any absorbing states.

Usage

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

Arguments

object	A chain_structure object.
...	Ignored.

Details

Columns are ordered for readability: identifiers first, classification second, dynamic per-state metrics last.

Value

A data.frame with one row per state. Columns described above.

summary.chain_structure_group

Cross-group comparison of chain_structure_group

Description

Produces a single tidy data.frame with one row per (group, state) combination, combining classification, persistence, sojourn, and – when applicable – stationary or mean-absorption-time columns. Useful for side-by-side reporting of chain_structure() across the members of a netobject_group.

Usage

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

Arguments

object	A chain_structure_group.
...	Ignored.

Value

A data.frame with columns group, state, classification, period, persistence, return_probability, sojourn_steps, plus stationary_probability if all groups are irreducible and mean_absorption_time if any group has absorbing states.

```
summary.cluster_choice
```

Summary Method for cluster_choice

Description

Summary Method for cluster_choice

Usage

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

Arguments

object	A cluster_choice object.
...	Unsupported. Supplying unused arguments raises an error.

Value

A data frame with the swept configurations, all metrics, and a best character column flagging the silhouette-max row.

```
summary.mcml
```

Summary Method for mcml

Description

Summary Method for mcml

Usage

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


Arguments

object	An mcml object.
...	Unsupported. Supplying unused arguments raises an error.

Value

A tidy data frame with one row per cluster and columns cluster, size, within_total, between_out, between_in. For undirected macro networks the in/out split is not meaningful, so between_out reports total incident weight and between_in is NA. The data frame is returned silently *without* printing the full object – call print(object) explicitly if you want the verbose dump.

Examples

```
seqs <- data.frame(V1 = c("A", "B", "C", "A"), V2 = c("B", "C", "A", "B"))
clusters <- list(G1 = c("A", "B"), G2 = c("C"))
cs <- build_mcml(seqs, clusters)
summary(cs)

seqs <- data.frame(
  T1 = c("A", "B", "A"), T2 = c("B", "C", "B"),
  T3 = c("C", "A", "C"), T4 = c("A", "B", "A")
)
clusters <- c("Alpha", "Beta", "Alpha")
cs <- build_mcml(seqs, clusters, type = "raw")
summary(cs)
```

summary.mcml_pc

*Summarize an MCML-PC Object***Description**

Summarize an MCML-PC Object

Usage

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

Arguments

object	An mcml_pc object.
...	Additional arguments (ignored).

Value

Tidy data frame with one row per macro edge (upper triangle), columns from, to, weight.

summary.mmm_compare	<i>Summary Method for mmm_compare</i>
---------------------	---------------------------------------

Description

Summary Method for mmm_compare

Usage

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

Arguments

object	An mmm_compare object (a data.frame subclass).
...	Unsupported. Supplying unused arguments raises an error.

Value

A tidy data frame with one row per k, plus a best character column flagging the minimum-BIC and minimum-ICL solutions.

summary.mosaic_analysis	<i>Summary method for mosaic_analysis objects</i>
-------------------------	---

Description

Summary method for mosaic_analysis objects

Usage

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

Arguments

object	A mosaic_analysis object.
...	Ignored.

Value

The tidy per-cell counts data.frame, with the stats row attached as an attribute.

`summary.nest_initial_probs`*Summary Method for Initial Probability Vectors*

Description

Summary Method for Initial Probability Vectors

Usage

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

Arguments

<code>object</code>	A <code>nest_initial_probs</code> named numeric vector returned by extract_initial_probs .
<code>...</code>	Additional arguments (ignored).

Value

A tidy data frame with columns `state` and `prob`, sorted by decreasing probability.

`summary.nest_transition_counts`*Summary Method for Transition Count Matrices*

Description

Summary Method for Transition Count Matrices

Usage

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

Arguments

<code>object</code>	A <code>nest_transition_counts</code> matrix returned by frequencies() .
<code>...</code>	Additional arguments (ignored).

Value

A tidy data frame with columns `from`, `to`, `count`, with one row per non-zero transition.

```
summary.nest_transition_matrix
```

Summary Method for Transition Matrices

Description

Summary Method for Transition Matrices

Usage

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

Arguments

object	A nest_transition_matrix returned by extract_transition_matrix .
...	Additional arguments (ignored).

Value

A tidy data frame with columns from, to, weight, with one row per non-zero entry.

```
summary.netobject
```

Network metrics for a netobject

Description

Computes node count, edge count, density, mean shortest-path distance, mean and SD of in/out strength, mean and SD of in/out degree, in/out degree centralization (Freeman), and reciprocity. Mirrors the metric set returned by `tna::summary.tna()` so a Nestimate netobject and the equivalent tna model report numerically identical descriptive metrics.

Usage

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

Arguments

object	A netobject (or cograph_network) object.
...	Ignored.

Value

A data.frame with columns metric and value, of class `c("summary.netobject", "data.frame")`.

`summary.netobject_group`*Network metrics for a netobject_group*

Description

Returns one summary per constituent network. With `combined = TRUE` (default) the per-group tables are joined into a single wide `data.frame` with one column per group; with `combined = FALSE` returns a named list.

Usage

```
## S3 method for class 'netobject_group'
summary(object, combined = TRUE, ...)
```

Arguments

<code>object</code>	A <code>netobject_group</code> .
<code>combined</code>	Logical. Combine into one wide <code>data.frame</code> ? Default <code>TRUE</code> .
<code>...</code>	Ignored.

Value

Either a `data.frame` (one column per group) or a named list of `summary.netobject` objects, of class `c("summary.netobject_group", ...)`.

`summary.net_association_rules`*Summary Method for net_association_rules*

Description

Summary Method for `net_association_rules`

Usage

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

Arguments

<code>object</code>	A <code>net_association_rules</code> object.
<code>...</code>	Additional arguments (ignored).

Value

A data frame summarizing the rules, invisibly.

Examples

```
trans <- list(c("A","B","C"), c("A","B"), c("B","C","D"), c("A","C","D"))
rules <- association_rules(trans, min_support = 0.3, min_confidence = 0.5,
                           min_lift = 0)
summary(rules)
```

summary.net_bayes	<i>Summary method for net_bayes</i>
-------------------	-------------------------------------

Description

Summary method for net_bayes

Usage

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

Arguments

object	A net_bayes object.
...	Additional arguments (ignored).

Value

A data frame with edge-level posterior differences and intervals.

Examples

```
s1 <- data.frame(V1 = c("A","B","C"), V2 = c("B","C","A"))
s2 <- data.frame(V1 = c("A","C","B"), V2 = c("C","B","A"))
b <- bayes_compare(build_network(s1, method = "relative"),
                   build_network(s2, method = "relative"),
                   draws = 500, seed = 1)
summary(b)
```

`summary.net_bayes_group`*Summary method for net_bayes_group*

Description

Summary method for net_bayes_group

Usage

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

Arguments

<code>object</code>	A net_bayes_group object.
<code>...</code>	Additional arguments (ignored).

Value

A combined data frame with a comparison column.

Examples

```
s <- data.frame(V1 = c("A", "B", "A", "C"), V2 = c("B", "C", "B", "A"),  
               grp = c("X", "X", "Y", "Y"))  
nets <- build_network(s, method = "relative", group = "grp")  
summary(bayes_compare(nets, draws = 200, seed = 1))
```

`summary.net_bootstrap` *Summary Method for net_bootstrap*

Description

Summary Method for net_bootstrap

Usage

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

Arguments

<code>object</code>	A net_bootstrap object.
<code>...</code>	Additional arguments (ignored).

Value

A data frame with edge-level bootstrap statistics.

Examples

```
net <- build_network(data.frame(V1 = c("A","B","C"), V2 = c("B","C","A")),
  method = "relative")
boot <- bootstrap_network(net, iter = 10)
summary(boot)

set.seed(1)
seqs <- data.frame(
  V1 = c("A","B","A","C","B"), V2 = c("B","C","B","A","C"),
  V3 = c("C","A","C","B","A")
)
net <- build_network(seqs, method = "relative")
boot <- bootstrap_network(net, iter = 20)
summary(boot)
```

summary.net_bootstrap_group

Summary Method for net_bootstrap_group

Description

Summary Method for net_bootstrap_group

Usage

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

Arguments

object	A net_bootstrap_group object.
...	Ignored.

Value

A data frame with group, edge, and bootstrap statistics columns.

Examples

```
seqs <- data.frame(V1 = c("A", "B", "A", "C"), V2 = c("B", "C", "C", "A"),
  V3 = c("C", "A", "B", "B"), grp = c("X", "X", "Y", "Y"))
nets <- build_network(seqs, method = "relative", group = "grp")
boot <- bootstrap_network(nets, iter = 10)
summary(boot)

set.seed(1)
seqs <- data.frame(
  V1 = c("A", "B", "A", "C", "B", "A"),
  V2 = c("B", "C", "B", "A", "C", "B"),
  V3 = c("C", "A", "C", "B", "A", "C"),
  grp = c("X", "X", "X", "Y", "Y", "Y")
)
nets <- build_network(seqs, method = "relative", group = "grp")
boot <- bootstrap_network(nets, iter = 20)
summary(boot)
```

summary.net_clustering

Summary Method for net_clustering

Description

Summary Method for net_clustering

Usage

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

Arguments

object	A net_clustering object.
...	Unsupported. Supplying unused arguments raises an error.

Value

The input object, invisibly.

Examples

```
seqs <- data.frame(V1 = c("A", "B", "C", "A", "B"), V2 = c("B", "C", "A", "B", "A"),
  V3 = c("C", "A", "B", "C", "B"))
cl <- build_clusters(seqs, k = 2)
summary(cl)

set.seed(1)
```

```
seqs <- data.frame(
  V1 = sample(c("A", "B", "C"), 20, TRUE),
  V2 = sample(c("A", "B", "C"), 20, TRUE),
  V3 = sample(c("A", "B", "C"), 20, TRUE)
)
cl <- build_clusters(seqs, k = 2)
summary(cl)
```

summary.net_gimme	<i>Summary Method for net_gimme</i>
-------------------	-------------------------------------

Description

Summary Method for net_gimme

Usage

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

Arguments

object	A net_gimme object.
...	Additional arguments (ignored).

Value

The input object, invisibly.

Examples

```
set.seed(1)
panel <- data.frame(
  id = rep(1:5, each = 20),
  t = rep(seq_len(20), 5),
  A = rnorm(100), B = rnorm(100), C = rnorm(100)
)
gm <- build_gimme(panel, vars = c("A", "B", "C"), id = "id", time = "t")
summary(gm)
```

summary.net_hon	<i>Summary Method for net_hon</i>
-----------------	-----------------------------------

Description

Summary Method for net_hon

Usage

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

Arguments

object	A net_hon object.
...	Additional arguments (ignored).

Value

The edge data.frame object\$edges (columns path, from, to, count, probability, from_order, to_order), returned visibly; the summary text is printed as a side effect.

Examples

```
seqs <- list(c("A","B","C","D"), c("A","B","C","A"), c("B","C","D","A"))  
hon <- build_hon(seqs, max_order = 2)  
summary(hon)
```

```
seqs <- data.frame(  
  V1 = c("A","B","C","A","B"),  
  V2 = c("B","C","A","B","C"),  
  V3 = c("C","A","B","C","A")  
)  
hon <- build_hon(seqs, max_order = 2L)  
summary(hon)
```

summary.net_honem	<i>Summary Method for net_honem</i>
-------------------	-------------------------------------

Description

Summary Method for net_honem

Usage

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

Arguments

- object A net_honem object.
- ... Additional arguments (ignored).

Value

A data.frame with one row per node: column node (node label) followed by dim1, dim2, ..., dimd embedding coordinates, returned visibly; the summary text is printed as a side effect.

Examples

```
seqs <- list(c("A","B","C","D"), c("A","B","C","A"), c("B","C","D","A"))  
hem <- build_honem(build_hon(seqs, max_order = 2), dim = 2)  
summary(hem)  
  
seqs <- list(c("A","B","C","D"), c("A","B","C","A"), c("B","C","D","A"))  
hon <- build_hon(seqs, max_order = 3)  
he <- build_honem(hon, dim = 2)  
summary(he)
```

summary.net_hypa	<i>Summary Method for net_hypa</i>
------------------	------------------------------------

Description

Summary Method for net_hypa

Usage

```
## S3 method for class 'net_hypa'
summary(
  object,
  n = 10L,
  type = c("all", "over", "under"),
  order_by = c("sig", "freq", "frequency", "ratio", "path"),
  ...
)
```

Arguments

object	A net_hypa object.
n	Integer. Maximum number of paths to display per category (default: 10).
type	Character. Which anomalies to show: "all" (default), "over", or "under".
order_by	Character. Ranking used within each anomaly direction: "sig" ranks by the active tail probability, "freq" by observed count, "ratio" by observed/expected ratio, and "path" alphabetically.
...	Additional arguments (ignored).

Value

A data frame with path, observed, expected, ratio, p_tail, and direction columns.

Examples

```
seqs <- list(c("A", "B", "C"), c("B", "C", "A"), c("A", "C", "B"), c("A", "B", "C"))
hyp <- build_hypa(seqs, k = 2)
summary(hyp)
```

```
seqs <- data.frame(
  V1 = c("A", "B", "C", "A", "B", "C", "A", "B", "C", "A"),
  V2 = c("B", "C", "A", "B", "C", "A", "B", "C", "A", "B"),
  V3 = c("C", "A", "B", "C", "A", "B", "C", "A", "B", "C"),
  V4 = c("A", "B", "C", "A", "B", "C", "A", "B", "C", "A")
)
hyp <- build_hypa(seqs, k = 2L)
summary(hyp)
summary(hyp, type = "over", n = 5)
```

```
summary.net_link_prediction
```

Summary Method for net_link_prediction

Description

Summary Method for net_link_prediction

Usage

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

Arguments

object	A net_link_prediction object.
...	Additional arguments (ignored).

Value

A data frame with per-method summary statistics, invisibly.

Examples

```
seqs <- data.frame(  
  V1 = sample(LETTERS[1:4], 30, TRUE),  
  V2 = sample(LETTERS[1:4], 30, TRUE),  
  V3 = sample(LETTERS[1:4], 30, TRUE)  
)  
net <- build_network(seqs, method = "relative")  
pred <- predict_links(net)  
summary(pred)
```

```
summary.net_markov_order
```

Summary Method for net_markov_order

Description

Summary Method for net_markov_order

Usage

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

Arguments

object A net_markov_order object.
 ... Ignored.

Value

The tidy test_table data.frame, with the selected optimal_order attached as an attribute.

Examples

```
# First-order Markov data: test should select order 1
set.seed(1)
states <- letters[1:4]
tm <- matrix(runif(16), 4, 4, dimnames = list(states, states))
tm <- tm / rowSums(tm)
seqs <- lapply(1:30, function(.) {
  s <- character(50); s[1] <- sample(states, 1)
  for (i in 2:50) s[i] <- sample(states, 1, prob = tm[s[i - 1], ])
  s
})
res <- markov_order_test(seqs, max_order = 3, n_perm = 300, seed = 1)
res$optimal_order
summary(res)
plot(res)
```

summary.net_mlvar	<i>Summary method for net_mlvar</i>
-------------------	-------------------------------------

Description

Summary method for net_mlvar

Usage

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

Arguments

object A net_mlvar object returned by [build_mlvar\(\)](#).
 ... Unused; present for S3 consistency.

Value

Invisibly returns object.

Examples

```
## Not run:
d <- simulate_data("mlvar", seed = 1)
fit <- build_mlvar(d, vars = attr(d, "vars"),
                  id = "id", day = "day", beep = "beep")
print(fit)
summary(fit)

## End(Not run)
```

summary.net_mmm	<i>Summary Method for net_mmm</i>
-----------------	-----------------------------------

Description

Summary Method for net_mmm

Usage

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

Arguments

object	A net_mmm object.
...	Unsupported. Supplying unused arguments raises an error.

Value

A per-component summary data.frame. The class and visibility depend on whether the model was fitted with covariates:

No covariates A plain data.frame with one row per component and columns component, prior, n_assigned, mean_posterior, avepp, returned *visibly* (so it auto-prints after the printed summary block).

With covariates A tidy_covariates/data.frame (the tidied covariate table, with the per-component stats attached), returned *invisibly*.

In both cases the printed summary (model fit, per-cluster transition matrices, optional covariate profiles) is emitted as a side effect.

Examples

```
seqs <- data.frame(V1 = sample(c("A", "B", "C"), 30, TRUE),
                   V2 = sample(c("A", "B", "C"), 30, TRUE))
mmm <- build_mmm(seqs, k = 2, n_starts = 1, max_iter = 10, seed = 1)
summary(mmm)

set.seed(1)
seqs <- data.frame(
  V1 = sample(c("A", "B", "C"), 30, TRUE),
  V2 = sample(c("A", "B", "C"), 30, TRUE),
  V3 = sample(c("A", "B", "C"), 30, TRUE)
)
mmm <- build_mmm(seqs, k = 2, n_starts = 5, seed = 1)
summary(mmm)
```

summary.net_mogen	<i>Summary Method for net_mogen</i>
-------------------	-------------------------------------

Description

Summary Method for net_mogen

Usage

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

Arguments

object	A net_mogen object.
...	Additional arguments (ignored).

Value

A per-order model-selection data.frame with columns order, layer_dof, cum_dof, loglik, aic, bic, best ("AIC"/"BIC"/"AIC+BIC" marker) and selected ("<--" on the chosen order), returned visibly; the summary text is printed as a side effect.

Examples

```
seqs <- list(c("A", "B", "C", "D"), c("A", "B", "C", "A"), c("B", "C", "D", "A"))
mg <- build_mogen(seqs, max_order = 2)
summary(mg)

seqs <- data.frame(
  V1 = c("A", "B", "C", "A", "B"),
```

```

V2 = c("B", "C", "A", "B", "C"),
V3 = c("C", "A", "B", "C", "A")
)
mog <- build_mogen(seqs, max_order = 2L)
summary(mog)

```

summary.net_nct

Summary Method for net_nct

Description

Returns a tidy data frame with one row per edge test. The global M (strength) and S (structure) statistics are attached as attributes.

Usage

```

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

```

Arguments

object	A net_nct object.
...	Ignored.

Value

A data frame with columns from, to, diff_observed, p_value, significant. Attributes m_stat and s_stat each hold a one-row data frame with observed and p_value.

Examples

```

## Not run:
set.seed(1)
x1 <- matrix(rnorm(200 * 5), 200, 5)
x2 <- matrix(rnorm(200 * 5), 200, 5)
colnames(x1) <- colnames(x2) <- paste0("V", 1:5)
res <- nct(x1, x2, iter = 100)
res$M$p_value
res$S$p_value

## End(Not run)

```

```
summary.net_path_dependence
```

Summary method for net_path_dependence

Description

Summary method for net_path_dependence

Usage

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

Arguments

object	A net_path_dependence object.
...	Ignored.

Value

A summary.net_path_dependence with the full sorted table and chain-level summaries.

```
summary.net_permutation
```

Summary Method for net_permutation

Description

Summary Method for net_permutation

Usage

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

Arguments

object	A net_permutation object.
...	Additional arguments (ignored).

Value

A data frame with edge-level permutation test results.

Examples

```

s1 <- data.frame(V1 = c("A","B","C"), V2 = c("B","C","A"))
s2 <- data.frame(V1 = c("A","C","B"), V2 = c("C","B","A"))
n1 <- build_network(s1, method = "relative")
n2 <- build_network(s2, method = "relative")
perm <- permutation(n1, n2, iter = 10)
summary(perm)

set.seed(1)
d1 <- data.frame(V1 = c("A","B","A"), V2 = c("B","C","B"),
                 V3 = c("C","A","C"))
d2 <- data.frame(V1 = c("C","A","C"), V2 = c("A","B","A"),
                 V3 = c("B","C","B"))
net1 <- build_network(d1, method = "relative")
net2 <- build_network(d2, method = "relative")
perm <- permutation(net1, net2, iter = 20, seed = 1)
summary(perm)

```

summary.net_permutation_group

Summary Method for net_permutation_group

Description

Returns a combined summary data frame across all groups.

Usage

```

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

```

Arguments

object	A net_permutation_group object.
...	Additional arguments (ignored).

Value

A data frame with group, edge, p_value, and sig columns.

Examples

```

s1 <- data.frame(V1 = c("A","B","A","C"), V2 = c("B","C","B","A"),
                 V3 = c("C","A","C","B"), grp = c("X","X","Y","Y"))
s2 <- data.frame(V1 = c("C","A","C","B"), V2 = c("A","B","A","C"),
                 V3 = c("B","C","B","A"), grp = c("X","X","Y","Y"))
nets1 <- build_network(s1, method = "relative", group = "grp")

```

```

nets2 <- build_network(s2, method = "relative", group = "grp")
perm <- permutation(nets1, nets2, iter = 10)
summary(perm)

set.seed(1)
s1 <- data.frame(V1 = c("A", "B", "A", "C"), V2 = c("B", "C", "B", "A"),
                 V3 = c("C", "A", "C", "B"), grp = c("X", "X", "Y", "Y"))
s2 <- data.frame(V1 = c("C", "A", "C", "B"), V2 = c("A", "B", "A", "C"),
                 V3 = c("B", "C", "B", "A"), grp = c("X", "X", "Y", "Y"))
nets1 <- build_network(s1, method = "relative", group = "grp")
nets2 <- build_network(s2, method = "relative", group = "grp")
perm <- permutation(nets1, nets2, iter = 20, seed = 1)
summary(perm)

```

summary.net_reliability

Summary Method for net_reliability

Description

Summary Method for net_reliability

Usage

```

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

```

Arguments

object	A net_reliability object.
...	Ignored.

Value

A tidy data frame with columns model, metric, mean, sd summarising the split-half iterations.

Examples

```

net <- build_network(data.frame(V1 = c("A", "B", "C", "A"),
                               V2 = c("B", "C", "A", "B")), method = "relative")
rel <- network_reliability(net, iter = 10)

seqs <- data.frame(
  V1 = sample(LETTERS[1:4], 30, TRUE), V2 = sample(LETTERS[1:4], 30, TRUE),
  V3 = sample(LETTERS[1:4], 30, TRUE), V4 = sample(LETTERS[1:4], 30, TRUE)
)
net <- build_network(seqs, method = "relative")
rel <- network_reliability(net, iter = 100, seed = 42)

```

```
print(rel)
```

```
summary.net_sequence_comparison
```

```
Summary Method for net_sequence_comparison
```

Description

Summary Method for net_sequence_comparison

Usage

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

Arguments

object	A net_sequence_comparison object.
...	Additional arguments (ignored).

Value

The patterns data.frame (tidy: one row per k-gram pattern, per group, with frequency and proportion columns; includes p-values when a permutation test was run).

Examples

```
seqs <- data.frame(  
  V1 = sample(LETTERS[1:4], 60, TRUE),  
  V2 = sample(LETTERS[1:4], 60, TRUE),  
  V3 = sample(LETTERS[1:4], 60, TRUE),  
  V4 = sample(LETTERS[1:4], 60, TRUE)  
)  
grp <- rep(c("X", "Y"), 30)  
net <- build_network(seqs, method = "relative")  
res <- sequence_compare(net, group = grp, sub = 2:3, test = "chisq")
```

`summary.net_stability` *Summary Method for net_stability*

Description

Returns the mean correlation at each drop proportion for each measure.

Usage

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

Arguments

<code>object</code>	A <code>net_stability</code> object.
<code>...</code>	Additional arguments (ignored).

Value

A data frame with columns `measure`, `drop_prop`, `mean_cor`, `sd_cor`, `prop_above`.

Examples

```
net <- build_network(data.frame(V1 = c("A","B","C","A"),  
  V2 = c("B","C","A","B")), method = "relative")  
cs <- centrality_stability(net, iter = 10, drop_prop = 0.3)  
summary(cs)  
  
set.seed(1)  
seqs <- data.frame(  
  V1 = sample(c("A","B","C"), 30, TRUE),  
  V2 = sample(c("A","B","C"), 30, TRUE),  
  V3 = sample(c("A","B","C"), 30, TRUE)  
)  
net <- build_network(seqs, method = "relative")  
stab <- centrality_stability(net, measures = c("InStrength","OutStrength"),  
  iter = 10)  
summary(stab)
```

```
summary.net_stability_group
  Summary Method for net_stability_group
```

Description

Per-network stability as a tidy data frame. Stacks `summary()` results for each network with a group column.

Usage

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

Arguments

<code>object</code>	A <code>net_stability_group</code> object.
<code>...</code>	Additional arguments (ignored).

Value

A data frame with columns `group`, `measure`, `drop_prop`, `mean_cor`, `sd_cor`, `prop_above`.

```
summary.net_transition_entropy
  Summary method for net_transition_entropy
```

Description

Returns a tidy per-state contribution table sorted by share of the chain-level entropy rate (largest first), so the dominant contributors to $h(P)$ are visible at a glance. Each row contains the stationary mass, the raw and normalised row entropy, the additive contribution $\pi_i H(P_{i\cdot})$, and that contribution as a percentage of $h(P)$.

Usage

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

Arguments

<code>object</code>	A <code>net_transition_entropy</code> object.
<code>...</code>	Ignored.

Value

A summary.net_transition_entropy containing

table tidy per-state data.frame, sorted by contribution_pct descending

chain tidy chain-level data.frame with raw and normalised $h(P)$, $H(\pi)$, redundancy, and ceiling

base logarithm base used

summary.net_vertex_bootstrap
<i>Summarize a Vertex Bootstrap Result</i>

Description

Summarize a Vertex Bootstrap Result

Usage

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

Arguments

object A net_vertex_bootstrap object.

... Additional arguments (ignored).

Value

The tidy summary data frame (one row per statistic).

summary.net_vertex_comparison
<i>Summarize a Two-Network Vertex-Bootstrap Comparison</i>

Description

Summarize a Two-Network Vertex-Bootstrap Comparison

Usage

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

Arguments

object A net_vertex_comparison object.

... Additional arguments (ignored).

Value

The tidy summary data frame (one row per statistic).

```
summary.wtna_boot_mixed
```

Summary Method for wtna_boot_mixed

Description

Summary Method for wtna_boot_mixed

Usage

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

Arguments

object	A wtna_boot_mixed object.
...	Additional arguments (ignored).

Value

A list with \$transition and \$cooccurrence summary data frames.

Examples

```
oh <- data.frame(A = c(1,0,1,0), B = c(0,1,0,1), C = c(1,1,0,0))
mixed <- wtna(oh, method = "both")
boot <- bootstrap_network(mixed, iter = 10)
summary(boot)

set.seed(1)
oh <- data.frame(
  A = c(1,0,1,0,1,0,1,0),
  B = c(0,1,0,1,0,1,0,1),
  C = c(1,1,0,0,1,1,0,0)
)
mixed <- wtna(oh, method = "both")
boot <- bootstrap_network(mixed, iter = 20)
summary(boot)
```

summary.wtna_perm_mixed
<i>Summary Method for wtna_perm_mixed</i>

Description

Summary Method for wtna_perm_mixed

Usage

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

Arguments

object A wtna_perm_mixed object.
... Additional arguments (ignored).

Value

A list with transition and co-occurrence permutation summaries.

trajectories	<i>Student Engagement Trajectories</i>
--------------	--

Description

Wide-format state sequences of student engagement over 15 weekly observations. Each row is one student; columns 1..15 hold the engagement state for that week. States: "Active", "Average", "Disengaged". Missing weeks are NA.

Usage

```
trajectories
```

Format

A character matrix with 138 rows and 15 columns. Entries are one of "Active", "Average", "Disengaged", or NA.

Examples

```
sequence_plot(trajectories, main = "Engagement trajectories")  
sequence_plot(trajectories, k = 3)  
sequence_plot(trajectories, type = "distribution")
```

transition_entropy	<i>Transition Entropy of a Markov Chain</i>
--------------------	---

Description

Computes per-state branching entropy, stationary entropy, and the chain-level entropy rate of a Markov transition process. The entropy rate is the Shannon-McMillan-Breiman per-step uncertainty of trajectories under the stationary distribution; it is the canonical information-theoretic summary of a transition matrix.

Usage

```
transition_entropy(x, base = 2, normalize = TRUE)
```

Arguments

<code>x</code>	A netobject, cograph_network, tna object, row-stochastic numeric transition matrix, or a wide sequence data.frame (rows = actors, columns = time-steps; a relative transition network is built automatically). Group dispatch on netobject_group.
<code>base</code>	Numeric. Logarithm base. 2 (default) for bits, exp(1) for nats, 10 for hartleys.
<code>normalize</code>	Logical. If TRUE (default), rows that do not sum to 1 are normalised automatically (with a warning).

Details

Convention $0 \log 0 := 0$ is applied, so absorbing or deterministic rows contribute zero per-row entropy. The chain need not be irreducible; π is computed from the eigendecomposition of $P^\top$ as elsewhere in the package. For non-ergodic chains the returned π is one stationary distribution among many - interpret with the help of [chain_structure](#).

The relation $h(P) \leq H(\pi)$ holds with equality iff successive states are independent. The deficit $H(\pi) - h(P)$ is reported as redundancy - a measure of how much memory the chain has at order 1.

Value

An object of class "net_transition_entropy" with:

row_entropy Named numeric vector, length n . Per-state branching entropy $H(P_{i\cdot}) = -\sum_j P_{ij} \log P_{ij}$.

row_entropy_norm Named numeric vector. row_entropy divided by the ceiling $\log_b n$ (in $[0, 1]$; all zeros when $n = 1$).

stationary Named numeric vector. Stationary distribution π .

stationary_entropy Scalar. $H(\pi) = -\sum_i \pi_i \log \pi_i$ - the entropy of π treated as an i.i.d. distribution. Upper bound on the entropy rate.

stationary_entropy_norm Scalar. stationary_entropy divided by the ceiling $\log_b n$.

entropy_rate Scalar. $h(P) = \sum_i \pi_i H(P_{i.})$ - the Shannon-McMillan-Breiman entropy rate.

entropy_rate_norm Scalar. entropy_rate divided by the ceiling $\log_b n$.

redundancy Scalar. $H(\pi) - h(P)$, the entropy deficit attributable to serial dependence; zero for an i.i.d. chain (rows of P all equal π).

redundancy_norm Scalar. The *relative* redundancy $(H(\pi) - h(P))/H(\pi)$ (the fraction of the stationary entropy removed by order-1 memory), **not** redundancy divided by $\log_b n$; 0 when $H(\pi) = 0$.

max_entropy Scalar. The normalising ceiling $\log_b n$.

base Logarithm base used.

states Character vector of state names.

References

Cover, T.M. & Thomas, J.A. (2006). *Elements of Information Theory*, 2nd ed., chapter 4. Wiley.

Shannon, C.E. (1948). A mathematical theory of communication. *Bell System Technical Journal*, 27, 379-423.

See Also

[markov_stability](#), [passage_time](#), [markov_order_test](#), [chain_structure](#)

Examples

```
net <- build_network(as.data.frame(trjectories), method = "relative")
te <- transition_entropy(net)
print(te)
summary(te)
plot(te)
```

validate_netobject	Validate a netobject / cograph_network against the shared schema
--------------------	--

Description

Enforces the structural contract that both Nestimate netobjects and **psychnet** objects must satisfy to be interchangeable across the package boundary. This is the single place that says what "a network object" means, so a drift on either side (a renamed field, a mistyped edge column) fails loudly here rather than mis-rendering three layers downstream.

Usage

```
validate_netobject(x)
```

Arguments

`x` An object expected to satisfy the `cograph_network` contract.

Details

The contract is deliberately the *shared* subset: the `$nodes` x/y layout columns and the `Nestimate` pipeline fields (`$data`, `$level`, ...) are not required, and `$edges` endpoints may be either integer node indices (`Nestimate`) or character labels (`psychnet`).

Value

Invisibly TRUE if `x` conforms; otherwise stops with the full list of violations.

See Also

[as_netobject](#)

Examples

```
net <- build_cor(data.frame(a = rnorm(50), b = rnorm(50), c = rnorm(50)))
validate_netobject(net)
```

<code>verify_simplicial</code>	<i>Verify Simplicial Complex Against igraph</i>
--------------------------------	---

Description

Cross-validates clique finding and Betti numbers against `igraph` and known topological invariants. Useful for testing.

Usage

```
verify_simplicial(mat, threshold = 0)
```

Arguments

`mat` A square adjacency matrix.
`threshold` Edge weight threshold.

Value

A list with `$cliques_match` (logical), `$n_simplices_ours`, `$n_simplices_igraph`, `$betti`, and `$euler`.

Examples

```
mat <- matrix(c(0,.6,.5,.6,0,.4,.5,.4,0), 3, 3)
colnames(mat) <- rownames(mat) <- c("A","B","C")
verify_simplicial(mat, threshold = 0.3)
```

Description

Non-parametric vertex bootstrap of a single observed network (Snijders & Borgatti 1999). Vertices are resampled with replacement and the weight matrix is rebuilt from the original entries of the resampled vertex pairs; network-level statistics computed on each replicate give bootstrap distributions, standard errors, and confidence intervals.

Unlike [bootstrap_network](#), which resamples the underlying cases (sequences or rows) and therefore requires the raw data stored in the netobject, the vertex bootstrap needs **only the weight matrix**. It works on any netobject - including data-less ones such as [build_mlvar](#) constituents or `as_tna(mcml)` elements - and on plain weight matrices. The two procedures answer different questions: the case bootstrap quantifies sampling-of-subjects uncertainty in the edge weights; the vertex bootstrap quantifies structural uncertainty of whole-network descriptives given the one network you observed.

Usage

```
vertex_bootstrap(
  x,
  iter = 1000L,
  ci_level = 0.05,
  ci_method = c("percentile", "basic"),
  statistics = NULL,
  statistic_fn = NULL,
  directed = NULL,
  seed = NULL
)
```

Arguments

<code>x</code>	A netobject (from build_network or any builder), a <code>cograph_network</code> , or a square numeric weight matrix.
<code>iter</code>	Integer. Number of bootstrap replicates (default 1000).
<code>ci_level</code>	Numeric. Significance level for the confidence intervals (default 0.05 for 95% CIs).
<code>ci_method</code>	Character. "percentile" (default) for empirical quantile intervals, or "basic" for intervals reflected around the observed value (Davison & Hinkley 1997, eq. 5.6).
<code>statistics</code>	Character vector selecting built-in statistics (see Details). Default: all applicable to the network's directedness.
<code>statistic_fn</code>	Optional named list of functions, each taking the weight matrix and returning a single numeric value. Computed alongside the built-ins.

directed	Logical or NULL. Directedness of the network. NULL (default) reads x\$directed when available, otherwise falls back to matrix symmetry.
seed	Integer or NULL. RNG seed for reproducibility.

Details

Each replicate draws n vertex indices with replacement and sets $W_b[i, j] = W[idx_i, idx_j]$. When the same original vertex is drawn for two different positions, the off-diagonal cell would be a structural self-pair; following Snijders & Borgatti, such cells are filled with the weight of a randomly chosen pair of distinct original vertices. Diagonal entries carry the original self-weight of the resampled vertex ($W[idx_i, idx_i]$) - self-loops are meaningful in transition networks and are never altered. For undirected networks the substitution is applied symmetrically so replicates stay symmetric.

Built-in statistics (all computed on the off-diagonal part of the weight matrix):

density Proportion of non-zero off-diagonal cells.

mean_weight Mean of the non-zero off-diagonal weights.

centralization Freeman-type strength centralization: $\text{sum}(\max(s) - s) / ((n - 1) * \max(s))$ where s is total node strength on absolute weights. 0 when all nodes have equal strength, approaching 1 for a star.

reciprocity Directed networks only. Weighted reciprocity $\text{sum}(\text{pmin}(|W|, |t(W)|)) / \text{sum}(|W|)$ over off-diagonal cells: the proportion of total weight that is reciprocated.

Value

An object of class "net_vertex_bootstrap" containing:

summary Tidy data frame, one row per statistic: statistic, observed, boot_mean, boot_sd, bias, ci_lower, ci_upper.

boot_stats iter x n_statistics matrix of replicate values.

observed Named vector of observed statistics.

iter, ci_level, ci_method, directed, n_nodes Configuration.

References

Snijders, T. A. B., & Borgatti, S. P. (1999). Non-parametric standard errors and tests for network statistics. *Connections*, 22(2), 161-170.

Davison, A. C., & Hinkley, D. V. (1997). *Bootstrap Methods and their Application*. Cambridge University Press.

See Also

[bootstrap_network](#) for case-resampling edge-weight inference, [centrality_stability](#) for case-dropping centrality stability.

Examples

```
seqs <- data.frame(
  T1 = c("plan", "code", "debug", "plan", "test", "code"),
  T2 = c("code", "debug", "code", "plan", "code", "test"),
  T3 = c("debug", "code", "plan", "code", "debug", "plan"),
  T4 = c("test", "plan", "test", "debug", "plan", "code")
)
net <- build_network(seqs, method = "relative")
vb <- vertex_bootstrap(net, iter = 100, seed = 1)
vb$summary

plot(vb)
```

vertex_compare

Compare Network-Level Statistics of Two Networks

Description

Snijders & Borgatti (1999) two-network test: each network's statistics get vertex-bootstrap standard errors, and each difference is tested with

$$z = (\hat{\theta}_x - \hat{\theta}_y) / \sqrt{SE_x^2 + SE_y^2}$$

against a standard normal reference. This is the comparison the vertex bootstrap was originally proposed for: deciding whether two observed networks differ in density, centralization, reciprocity, or any other whole-network descriptive.

Usage

```
vertex_compare(
  x,
  y,
  iter = 1000L,
  ci_level = 0.05,
  statistics = NULL,
  statistic_fn = NULL,
  directed = NULL,
  seed = NULL,
  labels = c("x", "y")
)
```

Arguments

x, y The two networks: `netobjects`, `cograph_networks`, square weight matrices, or precomputed `net_vertex_bootstrap` objects (then `iter`, `statistics`, `statistic_fn`, `directed`, and `seed` are ignored for that argument). The two sides must cover

	exactly the same statistics; a mismatch (e.g., a directed network's reciprocity against an undirected one, or precomputed objects built with different statistics selections) is an error, never a silent subset.
<code>iter</code>	Integer. Number of bootstrap replicates (default 1000).
<code>ci_level</code>	Numeric. Significance level for the confidence intervals (default 0.05 for 95% CIs).
<code>statistics</code>	Character vector selecting built-in statistics (see Details). Default: all applicable to the network's directedness.
<code>statistic_fn</code>	Optional named list of functions, each taking the weight matrix and returning a single numeric value. Computed alongside the built-ins.
<code>directed</code>	Logical or NULL. Directedness of the network. NULL (default) reads <code>x\$directed</code> when available, otherwise falls back to matrix symmetry.
<code>seed</code>	Integer or NULL. RNG seed for reproducibility.
<code>labels</code>	Character vector of length 2 naming the networks in the output (default <code>c("x", "y")</code>).

Value

An object of class "net_vertex_comparison" containing:

summary Tidy data frame, one row per statistic: `statistic`, the two observed values, `diff`, `se_diff`, `z`, `p_value`, and a normal-approximation confidence interval for the difference.

x, y The two `net_vertex_bootstrap` results.

labels, ci_level Configuration.

When both bootstrap SEs are zero (a statistic with no resampling variation in either network) `z` and `p_value` are NA.

References

Snijders, T. A. B., & Borgatti, S. P. (1999). Non-parametric standard errors and tests for network statistics. *Connections*, 22(2), 161-170.

See Also

[vertex_bootstrap](#), [nct](#) for the permutation-based comparison of edge-level structure when raw data are available, [permutation](#).

Examples

```
states <- c("plan", "code", "debug", "test")
s1 <- data.frame(
  T1 = rep(states, 5), T2 = rep(rev(states), 5),
  T3 = rep(states[c(2, 3, 4, 1)], 5)
)
s2 <- data.frame(
  T1 = rep(states[c(3, 1, 4, 2)], 5), T2 = rep(states, 5),
  T3 = rep(states[c(4, 3, 1, 2)], 5)
```

```
)  
net1 <- build_network(s1, method = "relative")  
net2 <- build_network(s2, method = "relative")  
cmp <- vertex_compare(net1, net2, iter = 100, seed = 1)  
cmp$summary
```

wide_to_long*Convert Wide Sequences to Long Format*

Description

Convert sequence data from wide format (one row per sequence, columns as time points) to long format (one row per action).

Usage

```
wide_to_long(  
  data,  
  id_col = NULL,  
  time_prefix = "V",  
  action_col = "Action",  
  time_col = "Time",  
  drop_na = TRUE  
)
```

Arguments

data	Data frame in wide format with sequences in rows.
id_col	Character. Name of the ID column, or NULL to auto-generate IDs. Default: NULL.
time_prefix	Character. Prefix for time point columns (e.g., "V" for V1, V2, ...). Default: "V".
action_col	Character. Name of the action column in output. Default: "Action".
time_col	Character. Name of the time column in output. Default: "Time".
drop_na	Logical. Whether to drop NA values. Default: TRUE.

Details

This function converts data from the format produced by `simulate_sequences()` to the long format used by many TNA functions and analyses.

Value

A data frame in long format with columns:

id Sequence identifier (integer).

Time Time point within the sequence (integer).

Action The action/state at that time point (character).

Any additional columns from the original data are preserved.

See Also

[long_to_wide](#) for the reverse conversion, [prepare_for_tna](#) for preparing data for TNA analysis.

Examples

```
wide_data <- data.frame(
  V1 = c("A", "B", "C"), V2 = c("B", "C", "A"), V3 = c("C", "A", "B")
)
long_data <- wide_to_long(wide_data)
head(long_data)
```

wtna

Window-based Transition Network Analysis

Description

Computes networks from one-hot (binary indicator) data using temporal windowing. Supports transition (directed), co-occurrence (undirected), or both network types.

Usage

```
wtna(
  data,
  method = c("transition", "cooccurrence", "both"),
  type = c("frequency", "relative"),
  codes = NULL,
  window_size = 3L,
  mode = c("non-overlapping", "overlapping"),
  actor = NULL
)
```

Arguments

<code>data</code>	Data frame with one-hot encoded columns (0/1 binary).
<code>method</code>	Character. Network type: "transition" (directed), "cooccurrence" (undirected), or "both" (returns list of two networks). Default: "transition".
<code>type</code>	Character. Output type: "frequency" (raw counts) or "relative" (row-normalized probabilities). Default: "frequency". Note that type = "relative" applied to method = "cooccurrence" produces an asymmetric matrix (conditional co-occurrence given row state), not a symmetric undirected weight matrix - use type = "frequency" if symmetric co-occurrence counts are required.
<code>codes</code>	Character vector or NULL. Names of the one-hot columns to use. If NULL, auto-detects binary columns. Default: NULL.
<code>window_size</code>	Integer (≥ 1). Number of consecutive rows to aggregate per window. Default: 3 (windowed pairwise between-window counting). Set window_size = 1 for ordinary consecutive ($t \rightarrow t+1$) transitions with no windowing.
<code>mode</code>	Character. Window mode: "non-overlapping" (fixed, separate windows) or "overlapping" (rolling, step = 1). Default: "non-overlapping".
<code>actor</code>	Character or NULL. Name of the actor/ID column for per-group computation. If NULL, treats all rows as one group. Default: NULL.

Details

Transitions: Uses `crossprod(X[-n, ], X[-1, ])` to count how often state i is active at time t AND state j at time $t+1$.

Co-occurrence: Uses `crossprod(X)` to count states that are simultaneously active in the same row.

Windowing: For `window_size > 1`, rows are aggregated into windows before computing networks. Non-overlapping windows are fixed, separate blocks; overlapping windows roll forward one row at a time. Within each window, any active indicator (1) in any row makes that state active for the window.

Per-actor: When `actor` is specified, networks are computed per group and summed.

Value

For `method = "transition"` or `"cooccurrence"`: a `netobject` (see [build_network](#)).

For `method = "both"`: a `wtna_mixed` object with elements `$transition` and `$cooccurrence`, each a `netobject`.

See Also

[build_network](#), [prepare_onehot](#)

Examples

```
oh <- matrix(c(1,0,0, 0,1,0, 0,0,1, 1,0,0), nrow = 4, byrow = TRUE,
             dimnames = list(NULL, c("A","B","C")))
w <- wtna(oh)
```

```
# Simple one-hot data
df <- data.frame(
  A = c(1, 0, 1, 0, 1),
  B = c(0, 1, 0, 1, 0),
  C = c(0, 0, 1, 0, 0)
)

# Transition network
net <- wtna(df)
print(net)

# Both networks
nets <- wtna(df, method = "both")
print(nets$transition)
print(nets$cooccurrence)

# With windowing
net <- wtna(df, window_size = 2, mode = "non-overlapping")
```

Index

* datasets

- chatgpt_srl, 76
- group_regulation_long, 108
- learning_activities, 112
- long-data, 115
- srl_strategies, 242
- trajectories, 275

- action_to_onehot, 7, 192
- actor_endpoints, 8
- actor_endpoints(), 123, 124
- ai_long(long-data), 115
- as.data.frame.net_cluster_diagnostics
(cluster_diagnostics), 79
- as.data.frame.state_freq(state_freq),
243
- as_htna, 11
- as_netdifference, 13, 20
- as_netobject, 14, 278
- as_networks, 15
- as_tna, 15, 16, 16
- association_rules, 9

- bayes_compare, 18, 74, 150
- betti_numbers, 20, 66
- bipartite_groups, 21
- bipartite_groups(), 77, 78, 109, 111, 112
- boot_glasso, 24
- bootstrap_network, 12, 22, 26, 52, 53, 63,
72–74, 132, 133, 139, 150, 279, 280
- bootstrap_network(), 55, 70
- bottleneck_distance, 27
- build_atna, 28
- build_clusters, 29, 57, 78, 79, 81–84, 100,
239, 240
- build_cna, 32, 97
- build_cor, 33
- build_ftna, 33
- build_gimme, 34
- build_glasso, 37
- build_hon, 37, 40
- build_honem, 40
- build_hypa, 41
- build_hypergraph, 43
- build_hypergraph(), 21, 22, 77, 78,
109–112
- build_ising, 45
- build_mcml, 11, 12, 46, 49, 53, 239, 240
- build_mcml_pc, 15, 16, 49, 114
- build_mlvar, 54, 279
- build_mlvar(), 89, 213, 263
- build_mmm, 56, 82, 83, 90, 239
- build_mogen, 58, 148
- build_network, 11, 12, 18, 20, 22–24, 26,
28–30, 32–34, 36, 37, 45, 46, 50, 58,
60, 64, 67, 71–73, 83, 84, 100, 101,
103, 132, 133, 143, 148–150, 188,
189, 191, 239, 279, 285
- build_network(), 45, 55, 75, 78, 118, 123,
124
- build_pcor, 64
- build_simplicial, 65
- build_simplicial(), 44, 45
- build_tna, 66

- casedrop_reliability, 67
- centrality_stability, 70, 280
- centrality_stability(), 69, 70
- certainty, 20, 24, 72
- chain_structure, 74, 276, 277
- chain_structure(), 9, 123, 124
- chatgpt_srl, 76
- clique_expansion, 77
- clique_expansion(), 109, 110, 112
- cluster_choice, 78
- cluster_diagnostics, 79, 79
- cluster_mmm, 81, 84, 169, 199, 215
- cluster_network, 81, 82, 83, 169, 199, 239
- cluster_summary, 16, 17, 46–48, 84
- coefs, 89

- coefs(), 55
- compare_mmm, 58, 78–80, 90
- compare_model, 91
- compare_model(), 118, 164
- compare_model.netobject_group, 93
- convert_sequence_format, 22, 51, 94, 107
- cooccurrence, 32, 95
- cooccurrence(), 77

- distribution_plot, 82, 98, 237, 239, 240

- estimate_network, 100, 233, 234
- euler_characteristic, 101
- evaluate_links, 102, 188
- extract_edges, 103, 104, 105
- extract_initial_probs, 104, 105, 251
- extract_transition_matrix, 103, 104, 105, 252

- frequencies, 95, 106, 251

- get_estimator, 107, 113, 234
- group_regulation_long, 108

- human_long (long-data), 115
- hypergraph_centrality, 109
- hypergraph_measures, 110
- hypergraph_measures(), 110

- learning_activities, 108, 112
- list_estimators, 63, 107, 113, 234, 235
- loading_stability, 52, 53, 114
- long-data, 115
- long_to_wide, 116, 191, 284

- magnitude_difference, 117
- mark_first_state, 122
- mark_terminal_state, 28, 32, 34, 62, 67, 123
- mark_terminal_state(), 9, 122, 123
- markov_order_test, 119, 147, 148, 277
- markov_stability, 121, 143, 277
- markov_stability(), 74, 75
- mogen_transitions, 124
- mosaic_analysis, 125, 129
- mosaic_plot, 126, 127, 183

- nct, 130, 282
- net_aggregate_weights, 133
- net_centrality, 15, 134, 160, 161
- net_deprune, 135, 139, 141
- net_edge_betweenness, 18, 136, 149, 164, 245
- net_hypergraph, 21, 77, 109, 110
- net_prune, 135, 136, 138, 140, 141
- net_pruning_details, 139, 140
- net_reprune, 135, 136, 139, 141
- netobject, 77
- network_reliability, 72, 74, 132

- p.adjust, 149, 236, 237
- p.adjust.methods, 42
- passage_time, 122, 142, 277
- passage_time(), 74, 75
- path_counts, 146
- path_dependence, 147
- pathways, 143
- permutation, 12, 18–20, 137, 148, 282
- persistence_landscape, 151
- persistent_homology, 66, 152
- plot, 166, 170
- plot.boot_glasso, 153
- plot.chain_structure, 154
- plot.cluster_choice, 155
- plot.magnitude_difference
 (magnitude_difference), 117
- plot.mcml_pc, 156
- plot.mmm_compare, 156
- plot.mosaic_analysis, 157
- plot.net_association_rules, 158
- plot.net_bayes, 159
- plot.net_casedrop_reliability
 (casedrop_reliability), 67
- plot.net_casedrop_reliability_group
 (casedrop_reliability), 67
- plot.net_centrality, 159
- plot.net_centrality_group, 160
- plot.net_cluster_diagnostics, 80, 163
- plot.net_clustering, 162, 163, 169
- plot.net_comparison, 163
- plot.net_edge_betweenness, 164
- plot.net_gimme, 165
- plot.net_honem, 166
- plot.net_markov_order, 167
- plot.net_markov_stability
 (markov_stability), 121
- plot.net_mmm, 163, 168
- plot.net_mmm_clustering, 163, 169
- plot.net_mogen, 170
- plot.net_mpt (passage_time), 142

plot.net_path_dependence, 171
 plot.net_reliability, 171
 plot.net_sequence_comparison, 172
 plot.net_stability, 173
 plot.net_transition_entropy, 174
 plot.net_vertex_bootstrap, 175
 plot.net_vertex_comparison, 175
 plot.pc_loading_stability, 176
 plot.persistence_landscape, 176
 plot.persistent_homology, 177
 plot.q_analysis, 178
 plot.simplicial_complex, 179
 plot.state_freq (state_freq), 243
 plot_mosaic, 130, 179
 plot_state_frequencies, 180, 181, 243
 predict_links, 11, 186
 predictability, 185
 prepare, 188
 prepare_for_tna, 116, 190, 284
 prepare_onehot, 189, 191, 285
 print.boot_glasso, 193
 print.chain_structure, 193
 print.chain_structure_group, 194
 print.cluster_choice, 194
 print.hypergraph_measures
 (hypergraph_measures), 110
 print.magnitude_difference
 (magnitude_difference), 117
 print.mcml, 195
 print.mcml_layer, 196
 print.mcml_pc, 196
 print.mmm_compare, 197
 print.mosaic_analysis, 197
 print.nestimate_data, 198
 print.net_association_rules, 201
 print.net_bayes, 202
 print.net_bayes_group, 203
 print.net_bootstrap, 24, 203
 print.net_bootstrap_group, 204
 print.net_casedrop_reliability
 (casedrop_reliability), 67
 print.net_casedrop_reliability_group
 (casedrop_reliability), 67
 print.net_certainty, 205
 print.net_cluster_diagnostics, 80, 207
 print.net_clustering, 199, 206, 207, 214,
 215
 print.net_gimme, 207
 print.net_hon, 208
 print.net_honem, 209
 print.net_hypa, 209
 print.net_hypergraph
 (build_hypergraph), 43
 print.net_link_prediction, 210
 print.net_markov_order, 211
 print.net_markov_order_group, 212
 print.net_markov_stability_group, 212
 print.net_mlvar, 213
 print.net_mmm, 199, 207, 214
 print.net_mmm_clustering, 215
 print.net_mogen, 215
 print.net_mpt_group, 216
 print.net_nct, 217
 print.net_path_dependence, 217
 print.net_permutation, 150, 218
 print.net_permutation_group, 219
 print.net_pruning_details, 220
 print.net_reliability, 220
 print.net_sequence_comparison, 221
 print.net_stability, 222
 print.net_stability_group, 223
 print.net_transition_entropy, 223
 print.net_transition_entropy_group,
 224
 print.net_vertex_bootstrap, 224
 print.net_vertex_comparison, 225
 print.netdifference
 (subtract_networks), 245
 print.netobject, 199
 print.netobject_group, 199
 print.netobject_ml, 200
 print.pc_loading_stability, 225
 print.persistence_landscape, 226
 print.persistent_homology, 226
 print.q_analysis, 227
 print.simplicial_complex, 228
 print.state_freq (state_freq), 243
 print.summary.net_casedrop_reliability_group
 (casedrop_reliability), 67
 print.summary.net_path_dependence, 228
 print.summary.net_transition_entropy,
 229
 print.summary_chain_structure, 229
 print.tidy_covariates, 230
 print.wtna_boot_mixed, 230
 print.wtna_mixed, 231

- `print.wtna_perm_mixed`, 232
- `q_analysis`, 66, 233
- `register_estimator`, 63, 107, 113, 233, 235
- `reliability`, 73
- `remove_estimator`, 234, 234
- `rename_models`, 235
- `sequence_compare`, 236
- `sequence_plot`, 82, 98–100, 237
- `simplicial_degree`, 66, 241
- `srl_strategies`, 108, 242
- `state_distribution`, 242
- `state_freq`, 243
- `state_frequencies`, 242, 244
- `subtract_networks`, 20, 245
- `summary.boot_glasso`, 246
- `summary.chain_structure`, 247
- `summary.chain_structure_group`, 247
- `summary.cluster_choice`, 248
- `summary.mcml`, 248
- `summary.mcml_pc`, 249
- `summary.mmm_compare`, 250
- `summary.mosaic_analysis`, 250
- `summary.nest_initial_probs`, 251
- `summary.nest_transition_counts`, 251
- `summary.nest_transition_matrix`, 252
- `summary.net_association_rules`, 253
- `summary.net_bayes`, 254
- `summary.net_bayes_group`, 255
- `summary.net_bootstrap`, 24, 255
- `summary.net_bootstrap_group`, 256
- `summary.net_casedrop_reliability`
(`casedrop_reliability`), 67
- `summary.net_casedrop_reliability_group`
(`casedrop_reliability`), 67
- `summary.net_clustering`, 257
- `summary.net_gimme`, 258
- `summary.net_hon`, 259
- `summary.net_honem`, 260
- `summary.net_hypa`, 260
- `summary.net_hypergraph`
(`build_hypergraph`), 43
- `summary.net_link_prediction`, 262
- `summary.net_markov_order`, 262
- `summary.net_mlvar`, 263
- `summary.net_mmm`, 264
- `summary.net_mogen`, 265
- `summary.net_mpt` (`passage_time`), 142
- `summary.net_nct`, 266
- `summary.net_path_dependence`, 267
- `summary.net_permutation`, 150, 267
- `summary.net_permutation_group`, 268
- `summary.net_reliability`, 269
- `summary.net_sequence_comparison`, 270
- `summary.net_stability`, 271
- `summary.net_stability_group`, 272
- `summary.net_transition_entropy`, 272
- `summary.net_vertex_bootstrap`, 273
- `summary.net_vertex_comparison`, 273
- `summary.netobject`, 252
- `summary.netobject_group`, 253
- `summary.wtna_boot_mixed`, 274
- `summary.wtna_perm_mixed`, 275
- `trajectories`, 275
- `transition_entropy`, 148, 276
- `validate_netobject`, 14, 277
- `verify_simplicial`, 278
- `vertex_bootstrap`, 52, 53, 279, 282
- `vertex_compare`, 52, 281
- `wide_to_long`, 116, 191, 283
- `wtna`, 284