Package: Nestimate (via r-universe)

June 3, 2026

Title Network Estimation, Bootstrap, and Higher-Order Analysis

Version 0.6.2

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>, psychological network methods (correlation, partial correlation, 'graphical lasso', 'Ising') Saqr, Beck, and Lopez-Pernas (2024) <doi: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

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

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

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

Config/testthat/edition 3

Depends R (>= 4.1.0)

LazyData true

VignetteBuilder knitr

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

Date/Publication 2026-06-03 19:27:10 UTC

RemoteUrl https://github.com/mohsaqr/Nestimate

RemoteRef HEAD

RemoteSha 61d0906d3f14e07f1af02cf0f9f354467f085f53

Contents

`action_to_onehot` *Convert Action Column to One-Hot Encoding*

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

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

---

actor_endpoints *Tidy per-actor endpoint summary of a wide-format sequence dataset.*

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

data	A wide-format matrix or data.frame where rows are actors and columns are time steps. Cells are state labels; `NA` represents missing observations. If `data` is a `data.frame`, non-character/-factor columns (e.g. an `id` column) are dropped via the `cols` argument.
cols	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()
```

---

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)
```

Excluded Clusters:

A within-cluster network is dropped only when row-normalisation would fail. Specifically, when the recorded `net_method` is `"relative"` (row-stochastic transitions) and any node in the cluster has zero outgoing weight, that cluster is excluded from `$clusters` and a `warning()` is emitted listing the dropped cluster names. For `net_method = "frequency"` (raw counts), a zero-row node is a legitimate sink and the cluster is retained. The macro / between-cluster network always includes every cluster regardless of the per-cluster drop decisions.

If a cluster you expect to see is missing from the returned `$clusters`, check the warning output and consider building with `type = "raw"` (which carries through to a frequency-method netobject and skips the drop) or inspect `rowSums(x$clusters[[cl]]$weights)`.

Value

A `cluster_tna` object (S3 class) containing:

between A tna object representing cluster-level transitions. Contains `$weights` (k x k transition matrix), `$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 `$weights` (n_i x n_i matrix), `$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
```

---

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 >= 4.

Metrics:

support P(A 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 and B) / (P(A) * P(B)). Values > 1 indicate positive association; < 1 indicate negative association.

conviction (1 - P(B)) / (1 - 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)
```

`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` *Hypergraph from bipartite group / event data*

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)
```

---

boot_glasso *Bootstrap for Regularized Partial Correlation Networks*

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 bootnet 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,
  nlambda = 100L,
  centrality = c("strength", "expected_influence", "betweenness", "closeness"),
  centrality_fn = NULL,
  cor_method = "pearson",
  ncores = 1L,
  seed = NULL
)
```

Arguments

x	A data frame, numeric matrix (observations x variables), or a netobject with method = "glasso".

iter	Integer. Number of nonparametric bootstrap iterations (default: 1000).
cs_iter	Integer. Number of case-dropping iterations per drop proportion (default: 500).
cs_drop	Numeric vector. Drop proportions for CS-coefficient computation (default: seq(0.1, 0.9, by = 0.1)).
alpha	Numeric. Significance level for CIs (default: 0.05).
gamma	Numeric. EBIC hyperparameter (default: 0.5).
nlambda	Integer. Number of lambda values in the regularization path (default: 100).
centrality	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 centrality_fn is supplied. Names that are not one of these four are valid only when a centrality_fn is supplied; that function is then responsible for returning them. Default: c("strength", "expected_influence", "betweenness", "closeness").
centrality_fn	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 rowSums, and "betweenness"/"closeness" via an internal Floyd-Warshall shortest-path routine. When provided, the function is called as centrality_fn(mat) and is used only for requested measures that are not one of the four built-ins; it should return a named list (e.g., list(my_metric = ...)).
cor_method	Character. Correlation method: "pearson" (default), "spearman", or "kendall".
ncores	Integer. Number of parallel cores for mclapply (default: 1, sequential).
seed	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")
```

bootstrap_network *Bootstrap a Network Estimate*

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")
)
```

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 ($<$ / $>$).

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

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)
```

`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 $D_1 \cup \Delta \to D_2 \cup \Delta$ 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 `.Machine$double.eps ^ 0.5`).

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 build_network. 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 nrow(data) - 1).
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: `c("*", "%")`.

Missing-value distance rule: after symbols are converted to `NA`, missing values are encoded as a single comparable sentinel state – *not* 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 `dissimilarity = "hamming"`. 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 `x$metadata` (and `x$data`) for `netobject` or `cograph_network` input.

character vector `c("Age", "Gender")`, same lookup.

formula `~ Age + Gender`, 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 `netobject` or `cograph_network` input, names are resolved against `$metadata` first and then non-state columns of `$data`, so a typical call looks like `build_clusters(net, 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 *Build a Co-occurrence Network (CNA)*

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 `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`, `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

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_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

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_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

`data`	A `data.frame` in long format with columns for person ID, time-varying variables, and optionally a time/beep column.
`vars`	Character vector of variable names to model.
`id`	Character string naming the person-ID column.

time	Character string naming the time/order column, or NULL. When provided, data is sorted by id then time before lagging.
ar	Logical. If TRUE (default), autoregressive paths (each variable predicting itself at lag 1) are included as fixed paths.
standardize	Logical. If TRUE (default FALSE), variables are standardized per person before estimation. Note: the returned coefficient network ($coefs, $psi, $temporal_avg, $contemporaneous_avg, $group_paths) is unaffected because Nestimate extracts the standardized lavaan solution (lavInspect(fit, "std")), which is invariant to input scaling. Only the scale-dependent $fit statistics (chisq, aic, bic) change.
groupcutoff	Numeric between 0 and 1. Proportion of individuals for whom a path must be significant to be added at group level. Default 0.75.
subcutoff	Numeric. Not used (reserved for future subgrouping).
paths	Character vector of lavaan-syntax paths to force into the model (e.g., "V2~V1lag"). Default NULL.
exogenous	Character vector of variable names to treat as exogenous. Default NULL.
hybrid	Logical. If TRUE, also searches residual covariances. Default FALSE.
rmsea_cutoff	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 x p matrix of group-level temporal (lagged) path counts – entry [i,j] = number of individuals with path j(t-1)->i(t).

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

coefs List of per-person p x 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 x 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 *Build a Graphical Lasso Network (EBICglasso)*

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 *Build a Higher-Order Network (HON)*

Description

Constructs a Higher-Order Network from sequential data, faithfully implementing the BuildHON algorithm (Xu, Wickramarathne & 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: • data.frame: rows are trajectories, columns are time steps. Trailing NAs are stripped. All non-NA values are coerced to character. • list: each element is a character (or coercible) vector representing one trajectory. • tna: a tna object with sequence data. Numeric state IDs are automatically converted to label names. • netobject: a netobject 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 min_freq 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., Wickramarathne, 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 `net_hon` 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 $by_order (named by order). The orders are sorted ascending internally, so the cograph_network slots ($weights, $edges, $adjacency, $xi, $nodes, $meta) always describe the network of the *lowest order that produced a layer* (a requested order with no edges is dropped from $by_order, $order and $k), regardless of the order in which the vector is given; $scores aggregates every built order.
alpha	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.
min_count	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.
p_adjust	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).
k	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 >= 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 *Higher-order hypergraph from a network's clique structure*

Description

Takes a network and produces a hypergraph by promoting k-cliques (k >= 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 >= 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 match.arg 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 >= 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 >= 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 >= 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: `list(A = c("N1","N2"), B = c("N3","N4"))`

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("N1","N2","N3"), group = c("A","A","B"))`

membership vector Character or numeric vector. Node names are extracted from the data. Example: `c("A","A","B","B")`

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 *both* 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 (`list(G1 = c("N1","N2"), ...)`) or a two-column data frame `data.frame(node, group)` instead.

NULL Auto-detect from `netobject$nodes` or `$node_groups` (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 *not* 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
```

---

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 *before* 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$temporal` Directed netobject for the `d x d` matrix of fixed-effect lagged coefficients. `$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 x d` partial-correlation network of within-person lmer residuals. `method = "mlvar_contemporaneous"`, `directed = FALSE`.

`fit$between` Undirected netobject for the `d x 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 `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`.

`params` Named list. Method-specific parameters passed to the estimator function (e.g. `list(gamma = 0.5)` for glasso, or `list(format = "wide")` 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 `weighted` and `concat` (and the low-level `begin_state` / `end_state`, of which `start` / `end` are the public form – see those arguments). Column-like entries in `params` (`action`, `id`, `id_col`, `time`, `session`, `order`, `codes`, and `group`) 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 data.frame `(name, label)`, a named character vector `c(name = "label")`, or

a named list. Rewrites `$nodes$label` and `dimnames(weights)`. 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_ml"` 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

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_pcor(srl_strategies)
```

---

build_simplicial *Build a Simplicial Complex*

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) > 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 *Build a Transition Network (TNA)*

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

x	A net_casedrop_reliability_group object.
iter	Integer. Iterations per drop proportion. Default 1000.
drop_prop	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.
threshold	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 net_casedrop_reliability_group.
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.
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 >= 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 igraph.

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,
  seed = NULL
)
```

Arguments

x	A netobject from build_network.
measures	Character vector. Centrality measures to assess. Built-in: "InStrength", "OutStrength", "Betweenness", "InCloseness", "OutCloseness", "Closeness". "Closeness" is defined only for undirected networks; "InCloseness"/"OutCloseness" only for directed networks (requesting the wrong one for the network's directedness is an error). Custom measures beyond these are valid only when a centrality_fn is supplied to resolve them. Default: c("InStrength", "OutStrength", "Betweenness").
iter	Integer. Number of bootstrap iterations per drop proportion (default: 1000).
drop_prop	Numeric vector. Proportions of cases to drop (default: seq(0.1, 0.9, by = 0.1)).
threshold	Numeric. Minimum correlation to consider stable (default: 0.7).
certainty	Numeric. Required proportion of iterations above threshold (default: 0.95).
method	Character. Correlation method: "pearson", "spearman", or "kendall" (default: "pearson").
centrality_fn	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 colSums/rowSums, and "Betweenness"/ "InCloseness"/"OutCloseness"/"Closeness" via an internal Floyd-Warshall shortest-path routine. When provided, the function is called as centrality_fn(mat) and is used only for requested measures that are not one of the six built-ins; it should return a named list (e.g., list(my_metric = ...)).
loops	Logical. If FALSE (default), self-loops (diagonal) are excluded from centrality computation. This does not modify the stored matrix.
seed	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)
```

---

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 x n` matrix. `[i, j] = P(ever reach j 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_transient x n_absorbing` matrix or `NULL` if no transient -> absorbing pathway exists. `[i, j] = P(eventual absorption in j | 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(trajectories), method = "relative")
cs  <- chain_structure(net)
print(cs)

summary(cs)
```

`chatgpt_srl` *ChatGPT Self-Regulated Learning Scale Scores*

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_nodes^2 * n_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 = incidence %*% 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 <= n - 1.
`dissimilarity`	Character vector of dissimilarity metrics. Use `"all"` to expand to every supported metric: `c("hamming", "osa", "lv", "dl", "lcs", "qgram", "cosine", "jaccard", "jw")`. 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

`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.

`cluster_by`
: Character. Accepted only as `"mmm"` (the default). Present so `cluster_mmm()` and `cluster_network()` 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` *Cluster data and build per-cluster networks in one step*

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")
```

---

cluster_summary *Cluster Summary Statistics*

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 x 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 x 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 x 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 `attr(result, "fits")` (a list of `net_mmm` 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 `build_mmm`.

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" $w/\max(\lvert w\rvert)$; preserves sign. "rank" Min-max of average ranks; ordinal scaling. "zscore" $(w - \bar{w})/s_w$; standard score. "robust" $(w - \mathrm{med}(w))/\mathrm{mad}(w)$; 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 $\lVert W\rVert_F = \sqrt{\sum w_{ij}^2}$; 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 *after* 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` *State Distribution Plot Over Time*

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,
  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: data.frame, matrix, netobject, net_clustering, netobject_group, net_mmm, or tna. When clustering info is available, one panel is drawn per cluster.
group	Optional grouping vector (length nrow(x)) producing one panel per group. Ignored if x is a net_clustering.
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 na_color.
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 time_label.
y_label	Y-axis label. Defaults to "Proportion" or "Count" based on scale.
ylab	Alias for y_label.
tick	Show every Nth x-axis label. NULL = auto.
ncol, nrow	Facet grid dimensions. NULL = auto: ncol = ceiling(sqrt(G)), nrow = ceiling(G / ncol). Ignored when combined = FALSE.
combined	When TRUE (default), groups are arranged on one figure via graphics::layout(). 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 (fig.show = "asis") or to save each as a separate file. Single-group calls (G == 1) 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(trajectories))
```

`estimate_network` *Estimate a Network (Deprecated)*

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. list(gamma = 0.5) for glasso, or list(format = "wide") 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 weighted and concat (and the low-level begin_state / end_state, of which start / end are the public form – see those arguments). Column-like entries in params (action, id, id_col, time, session, order, codes, and group) 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.
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` *Evaluate Link Predictions Against Known Edges*

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` *Extract Edge List with Weights*

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

`model`	A TNA model object or a list containing a 'weights' element.
`type`	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)
```

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 igraph::eigen_centrality()'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 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 intersect e_j|`. Diagonal is zero.

`overlap_coefficient` `m` x `m`: `|e_i 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 and e_j| / |e_i union e_j|`.

Global (scalars)

`density` For k-uniform hypergraphs: `m / choose(n, k)`. For mixed sizes: `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 *List All Registered Estimators*

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()
```

long-data *Human-AI Vibe Coding Interaction Data (Long Format)*

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

data	Data frame in long format.
id_col	Character. Name of the column identifying sequences. Default: "Actor".
time_col	Character. Name of the column identifying time points. Default: "Time".
action_col	Character. Name of the column containing actions/states. Default: "Action".
time_prefix	Character. Prefix for time point columns in output. Default: "V".
fill_na	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` *Magnitude difference between the frequency and probability views*

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 build_network(); "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
```

---

mark_first_state *Mark leading-NA cells with an explicit state label.*

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

---

mark_terminal_state *Mark terminal-NA cells with an explicit state label.*

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[state, 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)
```

---

markov_order_test *Test the Markov order of a sequential process*

Description

Principled test of whether a categorical sequence is best described as a k-th order Markov chain. At each order $k = 1, \ldots,$ `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.
`parallel`	Logical. Use `parallel::mclapply` for permutations. Default `FALSE` (set `TRUE` only on Unix-like systems).
`n_cores`	Integer. Cores for parallel execution. Default 2.
`seed`	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)
```

---

markov_stability *Markov Stability Analysis*

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(trajectories), method = "relative")
ms  <- markov_stability(net)
print(ms)

plot(ms)
```

---

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

x	A net_mogen object from build_mogen().
order	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.
min_count	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(to | 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_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,
  ...
)

## 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,
  ...
)

## S3 method for class 'mcml'
```

```
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,
  ...
)

## 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,
  ...
)

## 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,
  ...
)

## S3 method for class 'matrix'
```

```
mosaic_plot(x, ...)
```

Arguments

x	One of the four data-bearing Nestimate classes: netobject (single mosaic of $weights), netobject_group (one panel per group), mcml (between-cluster mosaic by default; per-cluster panels with level = "within"), or htna (single mosaic of $weights; htna inherits netobject so the geometry matches). Also accepts a contingency table or plain numeric matrix for ad-hoc plotting.
...	Ignored.
xlab, ylab	Axis labels. NULL (default) draws no axis title. Pass any string to add one.
range	Numeric of length 2 giving the lower and upper colour-scale limits for the standardized residual. NULL (default) auto-fits the limits to the symmetric range c(-M, M) where M = max(|stdres|), so no signal is squished. Pass an explicit range (e.g. c(-4, 4) for tna-style display, c(-6, 6) for moderate clipping) to clamp the colour scale.
top_angle, left_angle	Rotation in degrees for the top (x) and left (y) tick labels. NULL (default) uses the auto rule 90 if n_levels > 3 else 0 on each axis. Pass any numeric to override (e.g. top_angle = 45, left_angle = 0).
residuals	One of "permutation" (default) or "asymptotic". "permutation" computes empirical-null z-scores by shuffling one variable's labels against the other for n_perm draws and reporting (O - mean_perm) / sd_perm per cell. Robust on sparse tables. "asymptotic" returns stats::chisq.test()$stdres (the closed-form (O - E) / sqrt(E*(1 - p_row)*(1 - p_col)) that vcd and tna use).
n_perm	Number of permutations when residuals = "permutation". Default 500; use >= 1000 for stable tail estimates.
seed	Optional integer seed for the permutation RNG. Use for reproducible plots; ignored when residuals = "asymptotic".
values	Logical. When TRUE, overlay each cell's standardized residual as a numeric label (one decimal). Text colour switches to white on saturated cells (|stdres| > 1.5) and dark grey otherwise. Default FALSE – the colour bar legend already conveys the sign and magnitude.
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 /

htna 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 *Network Comparison Test*

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

data1	A numeric matrix or data.frame of observations from group 1.
data2	A numeric matrix or data.frame of observations from group 2. Same number of columns as data1.
iter	Integer. Number of permutation iterations. Default 1000.

gamma	EBIC tuning parameter for glasso. Default 0.5.
paired	Logical. If TRUE, perform a paired permutation (within-subject swap). Default FALSE.
abs	Logical. If TRUE, compute global strength on absolute edge weights. Default TRUE.
weighted	Logical. If TRUE, use weighted networks for the tests. If FALSE, binarize before computing statistics. Default TRUE.
p_adjust	P-value adjustment method for the per-edge tests (any method in stats::p.adjust.methods). 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)
```

`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"` (`sum(w) / n_possible`) and `"mean"` (`sum(w) / number of non-zero edges`) return the *same* 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 `sum(w) / 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, or cograph_network. For directed networks the default measures are InStrength, OutStrength, and Betweenness. For undirected networks the defaults are Closeness and Betweenness.

Usage

```
net_centrality(x, measures = NULL, loops = FALSE, centrality_fn = NULL, ...)
```

Arguments

x	A netobject, netobject_group, or cograph_network.
measures	Character vector. Centrality measures to compute. Built-in: "InStrength", "OutStrength", "Betweenness", "InCloseness", "OutCloseness", "Closeness". "Closeness" is defined only for undirected networks; "InCloseness"/"OutCloseness" only for directed networks (requesting the wrong one for the network's directedness is an error). Default depends on directedness.
loops	Logical. Include self-loops (diagonal) in computation? Default: FALSE.
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 data frame with node names as rows and centrality measures as columns. For a netobject_group: a named list of such data frames (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")
net_centrality(net)
```

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)
```

---

passage_time *Mean First Passage Times*

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(trajectories), method = "relative")
pt  <- passage_time(net)
print(pt)

plot(pt)
```

`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` *Per-Context Path Dependence at Order k*

Description

Diagnoses where a chain's order-1 Markov assumption fails by comparing, for each order-k context $(s_1, \ldots, s_{k-1})$, the empirical next-state distribution $P(s_k \mid s_1, \ldots, s_{k-1})$ 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 $c = (s_1, \ldots, s_{k-1})$ 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 $D_{KL}(P_k \,\|\, P_1)$ - 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 (= $D_{KL}(P_k \| P_1)$), 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)
```

---

pathways *Extract Pathways from Higher-Order Network Objects*

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

x	A higher-order network object (net_hon, net_hypa, or net_mogen).
...	Additional arguments.
min_count	Integer. Minimum transition count to include (default: 1).
min_prob	Numeric. Minimum transition probability to include (default: 0).
top	Integer or NULL. Return only the top N pathways ranked by count (default: NULL = all).
order	Integer or NULL. Markov order to extract. Default: optimal order from model selection.
type	Character. Which anomalies to include: "all" (default), "over", or "under".
ho_method	Character. Higher-order method: "hon" (default) or "hypa".
min_lift	Numeric or NULL. Additional lift filter applied on top of the object's original threshold (default: NULL).
min_confidence	Numeric or NULL. Additional confidence filter (default: NULL).
method	Character or NULL. Which prediction method to use. Default: first method in the object.
evidence	Logical. If TRUE, include common neighbor evidence nodes in each pathway. Default: TRUE.
max_evidence	Integer. Maximum number of evidence nodes per pathway (default: 3).

Value

A character vector of pathway strings in arrow notation (e.g. "A B -> C"), suitable for cograph::plot_simplicial().

A character vector of pathway strings.

A character vector of pathway strings.

A character vector of pathway strings.

A character vector of pathway strings.

A character vector of pathway strings.

A character vector of pathway strings.

Methods (by class)

- pathways(net_hon): Extract higher-order pathways from HON
- pathways(net_hypa): Extract anomalous pathways from HYPA
- pathways(netobject): Extract pathways from a netobject
  Builds a Higher-Order Network (HON) from the netobject's sequence data and returns the higher-order pathways. Requires that the netobject was built from sequence data (has $data).
- pathways(net_association_rules): Extract pathways from association rules
  Converts association rules {A, B} => {C} into pathway strings "A B -> C" suitable for cograph::plot_simplicial(). Antecedent items become source nodes; consequent items become the target.
- pathways(net_link_prediction): Extract pathways from link predictions
  Converts predicted links into pathway strings for cograph::plot_simplicial(). When evidence = TRUE (default), each predicted edge A -> B is enriched with common neighbors that structurally support the prediction, producing "A cn1 cn2 -> B".
- pathways(net_mogen): Extract transition pathways from MOGen

Examples

```
seqs <- list(c("A","B","C","D"), c("A","B","C","A"))
hon <- build_hon(seqs, max_order = 3)
pw <- pathways(hon)


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 = 0.3,
                           min_lift = 0)
pathways(rules)

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)
```

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.

Usage

```
permutation(
  x,
  y = NULL,
  iter = 1000L,
  alpha = 0.05,
  paired = FALSE,
  adjust = "none",
  nlambda = 50L,
  seed = NULL
)
```

Arguments

x	A netobject (from build_network).
y	A netobject (from build_network). 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".
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.

iter Number of permutation iterations.

alpha Significance level used.

paired Whether paired permutation was used.

adjust p-value adjustment method used.

See Also

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 = 0` 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) > 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 *Plot Method for boot_glasso*

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

x	A chain_structure object.
show_values	Logical. If TRUE (default), prints the numeric probability inside each cell. Set FALSE for large state spaces (n > 10) where labels overlap.
digits	Integer. Decimal places for in-cell labels.
...	Ignored.

Details

Cell colour encodes P(ever reach j | start at i). The diagonal uses the return-time convention (P(return to j in >= 1 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 categoricals.

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.

`plot.mmm_compare` *Plot Method for mmm_compare*

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.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_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_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

x	A net_clustering object.
type	Character. Plot type: "silhouette" (per-observation silhouette bars), "mds" (2D MDS projection), or "heatmap" (distance matrix heatmap ordered by cluster). Default: "silhouette".
combined	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.
...	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_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 compare_model().
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_gimme *Plot Method for net_gimme*

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 stitched side-by-side via gridExtra::grid.arrange. When FALSE, returns a named list (ic, permutation) of ggplots. Ignored when panel != "both".
...	Ignored.

Value

A ggplot (single panel), a gridExtra-arranged grob (both, combined), or a named list of two ggplots (both, not combined).

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 *Plot Method for net_mogen*

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("pyramid", "heatmap"),
  sort = c("statistic", "frequency"),
  alpha = 0.05,
  show_residuals = FALSE,
  ...
)
```

Arguments

`x`	A `net_sequence_comparison` object.
`top_n`	Integer. Show top N patterns. Default: 10.
`style`	Character. `"pyramid"` (default) or `"heatmap"`.
`sort`	Character. `"statistic"` (default) ranks patterns by test statistic or residual magnitude. `"frequency"` ranks by total occurrence count across all groups.
`alpha`	Numeric. Significance threshold for p-value display in the pyramid: patterns with `p_value < alpha` are starred and drawn in bold dark text, the rest stay plain grey. Default: 0.05.
`show_residuals`	Logical. If `TRUE`, print the standardized residual value inside each pyramid bar. Default: `FALSE`. Ignored for the heatmap (which always shows residuals).
`...`	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 *Plot Method for net_stability*

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.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 gridExtra::arrangeGrob. 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` *Plot Q-Analysis*

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 `gridExtra::arrangeGrob`. When `FALSE`, returns a named list (`q_vector`, `structure_vector`) 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 gridExtra::arrangeGrob and drawn. When FALSE, returns a named list of the four ggplots (f_vector, betti, degree, degree_heatmap) 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

data	A data.frame in long form. Must contain the columns named in x, y, and weight.
x	Column name for the X (column) variable.
y	Column name for the Y (segment) variable.
weight	Column name for the cell weight (e.g. count).
fill	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).
colors	Optional character vector of fill colors. When fill = "y", length must be at least the number of distinct y levels. Defaults to recycled Okabe-Ito.
show_labels	If TRUE, draw within-segment percentage labels.
label_size	Numeric size for segment labels.
x_label, y_label	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 base::abbreviate() (which extends the truncation as needed to keep names unique after collision); a positive integer sets the target minimum length explicitly (e.g. abbreviate = 4). Affects tile labels, legend, and the returned $table.
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 `mcml` 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 `mcml`, 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))
}
```

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`.
`top_n`	Integer or NULL. Return only the top N predictions per method. Default: `NULL` (all pairs).
`exclude_existing`	Logical. If `TRUE`, exclude node pairs that already have an edge. Default: `TRUE`.
`include_self`	Logical. If `TRUE`, include self-loop predictions. Default: `FALSE`.
`katz_damping`	Numeric or NULL. Attenuation factor for Katz index. If NULL, auto-computed as `0.9 / 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/degree(z)`. Penalizes hubs, rewards rare shared connections.

adamic_adar Adamic & Adar (2003). Like resource allocation but weights by `1/log(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 - 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)
```

`predictability` *Compute Node Predictability*

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)
```

---

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

`data`	Data frame with event log columns.
`actor`	Character or character vector. Column name(s) identifying who performed the action (e.g. `"student"` or `c("student", "group")`). If missing, all data is treated as one actor.
`action`	Character. Column name containing the action/state/code.
`time`	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.
`order`	Character or NULL. Column name for tie-breaking when timestamps are identical. If NULL, original row order is used. Default: NULL.
`session`	Character, character vector, or NULL. Column name(s) for explicit session grouping (e.g. `"course"` or `c("course", "semester")`). When combined with `time`, sessions are further split by time gaps. Default: NULL.
`time_threshold`	Numeric. Maximum gap in seconds between consecutive events before a new session starts. Only used when `time` is provided. Default: 900 (15 minutes).
`custom_format`	Character or NULL. Custom `strptime` format string for parsing timestamps. Default: NULL (auto-detect).
`is_unix_time`	Logical. If TRUE, treat numeric time values as Unix timestamps. Default: FALSE (auto-detected for numeric columns).
`unix_time_unit`	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` *Prepare Data for TNA Analysis*

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

`data`	Data frame containing sequence data.
`type`	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.
`state_names`	Character vector. Expected state names, or NULL to extract from data. Default: NULL.
`id_col`	Character. Name of ID column for long format data. Default: "Actor".
`time_col`	Character. Name of time column for long format data. Default: "Time".
`action_col`	Character. Name of action column for long format data. Default: "Action".
`validate`	Logical. Whether to validate that all actions are in state_names. 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

data	Data frame with binary (0/1) indicator columns.
cols	Character vector. Names of the one-hot columns to use.
actor	Character or NULL. Name of the actor/ID column. If NULL, all rows are treated as a single sequence. Default: NULL.
session	Character or NULL. Name of the session column for sub-grouping within actors. Default: NULL.
interval	Integer or NULL. Number of rows per time point in the output. If NULL, all rows become a single time point group. Default: NULL.
window_size	Integer (>= 1). Number of consecutive rows to aggregate into each window. Default: 3. Set window_size = 1 for no windowing (each row is its own time point).
window_type	Character. "non-overlapping" (fixed, separate windows) or "overlapping" (rolling, step = 1). Default: "non-overlapping".
aggregate	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 *Print Method for boot_glasso*

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 *Print method for* chain_structure.

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 *Print method for* chain_structure_group.

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 *Print Method for cluster_choice*

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 *Print Method for mcml*

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 *Print Method for an mcml Layer*

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.mmm_compare *Print Method for mmm_compare*

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.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.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_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_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_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_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 *Print Method for net_honem*

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 *Print Method for net_hypa*

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

x	A net_markov_order_group (named list of net_markov_order results, one per group).
...	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 *Print method for net_mlvar*

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

x	A net_mmm_clustering object.
digits	Integer. Decimal places for floating-point statistics. Default 3.
...	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 *Print Method for net_nct*

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` *Print method for* `net_path_dependence`

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_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.netobject *Print Method for Network Object*

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 *Print Method for Group Network Object*

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.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 *Print Q-analysis results*

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

x	A summary_chain_structure object.
...	Forwarded to print.data.frame.

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

x	A tidy_covariates data.frame.
...	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 *Print Method for wtna_mixed*

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

x	A wtna_perm_mixed object.
...	Additional arguments (ignored).

Value

The input object, invisibly.

---

q_analysis *Q-Analysis*

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) <- rownames(mat) <- c("A","B","C")
sc <- build_simplicial(mat, threshold = 0.3)
q_analysis(sc)
```

register_estimator *Register a Network Estimator*

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 *Remove a Registered Estimator*

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 *Rename the models of a* netobject_group

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

x	A netobject_group (or any object inheriting from it, such as net_mlvar).
new_names	A character vector of new names. Must have the same length as x, 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 underpowered 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,
  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 `$data`. **net_clustering** From `build_clusters`. Uses `$data`, `$assignments` for grouping, and `$distance` for dendrogram. **netobject_group** From `cluster_network` or `build_network` on a clustering. Extracts data and assignments from `attr(, "clustering")`. **net_mmm** From `build_mmm`. Uses `$models[[1]]$data` and `$assignments`. **tna** From the tna package. Decodes integer-encoded sequences.
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 `build_clusters` distance (`"hamming"`, `"osa"`, `"lv"`, `"dl"`, `"qgram"`, `"cosine"`, `"jaccard"`, `"jw"`).
tree	Optional `hclust`/`dendrogram`/`agnes` object to supply row ordering (heatmap only; overrides `sort`).
group	Optional grouping vector (length `nrow(x)`) producing one facet per group. Index/distribution only. Ignored for heatmap.
scale, geom, na	Passed to `distribution_plot` when `type = "distribution"`.
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 `type = "heatmap"`, cuts the dendrogram into k clusters and draws thin horizontal separators between them in the carpet. Ignored when there is no dendrogram (e.g. `sort = "start"`) 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 grDevices::dev.new(). In knitr chunks use the fig.width / fig.height 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 graphics::layout(). When FALSE, each group is drawn on its own page (one full-size figure per group, with its own legend). Single-group calls (G == 1) 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 [0.65, 1.2]).
legend_title	Optional legend title.
legend_ncol	Number of legend columns.
legend_border	Swatch border colour.
legend_bty	"n" or "o".

Value

Invisibly, a list describing the plot (shape depends on type).

See Also

distribution_plot, build_clusters

Examples

```
sequence_plot(trajectories)
sequence_plot(trajectories, type = "index")
sequence_plot(trajectories, type = "distribution")
```

simplicial_degree *Simplicial Degree*

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 *Self-Regulated Learning Strategy Frequencies*

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 *Per-Class State Distribution as a Tidy Data Frame*

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 *Compute State Frequencies from Trajectory Data*

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)
```

---

summary.boot_glasso *Summary Method for boot_glasso*

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.mmm_compare *Summary Method for mmm_compare*

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.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

object	A nest_initial_probs named numeric vector returned by extract_initial_probs.
...	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

`object`	A `nest_transition_counts` matrix returned by `frequencies()`.
`...`	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.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

object	A net_association_rules object.
...	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_bootstrap *Summary Method for net_bootstrap*

Description

Summary Method for net_bootstrap

Usage

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

Arguments

object	A net_bootstrap object.
...	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` *Summary Method for net_gimme*

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 *Summary Method for net_hon*

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 *Summary Method for net_honem*

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, ..., dim*d* 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 *Summary Method for net_hypa*

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")
)
hypa <- build_hypa(seqs, k = 2L)
summary(hypa)
summary(hypa, 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 *Summary method for net_mlvar*

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` *Summary Method for net_mmm*

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 *Summary Method for net_mogen*

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

object	A net_stability object.
...	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

`object`	A `net_stability_group` object.
`...`	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

`object`	A `net_transition_entropy` object.
`...`	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.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

object	A netobject_group.
combined	Logical. Combine into one wide data.frame? Default TRUE.
...	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.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` *Summary Method for wtna_perm_mixed*

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` *Student Engagement Trajectories*

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` *Transition Entropy of a Markov Chain*

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

`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). Group dispatch on `netobject_group`.
`base`	Numeric. Logarithm base. `2` (default) for bits, `exp(1)` for nats, `10` for hartleys.
`normalize`	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. $h(P) = \sum_i \pi_i H(P_{i\cdot})$ - 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(trajectories), method = "relative")
te  <- transition_entropy(net)
print(te)
summary(te)
plot(te)
```

verify_simplicial *Verify Simplicial Complex Against igraph*

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)
```

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

data	Data frame with one-hot encoded columns (0/1 binary).
method	Character. Network type: "transition" (directed), "cooccurrence" (undirected), or "both" (returns list of two networks). Default: "transition".
type	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.
codes	Character vector or NULL. Names of the one-hot columns to use. If NULL, auto-detects binary columns. Default: NULL.
window_size	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 -> t+1) transitions with no windowing.
mode	Character. Window mode: "non-overlapping" (fixed, separate windows) or "overlapping" (rolling, step = 1). Default: "non-overlapping".
actor	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