Package 'dynamicmultiplex'

August 7, 2026

Title Community Detection for Evolving Multiplex Networks

Version 1.1.0

Description
Multiplex temporal community detection with customizable interlayer coupling. Runs Louvain or Leiden community detection on each network layer and constructs interlayer ties using Jaccard similarity, overlap coefficient, node-strength weighted variants, or direct node identity links, and also provides a two-stage snapshot-and-match tracker that aligns independently detected per-layer communities across time with the Hungarian assignment algorithm. Supports user-specified layer connectivity via the layer_links argument, enabling adjacent-only temporal coupling that avoids the long-range pooling problem in standard multislice approaches.

License MIT + file LICENSE

Encoding UTF-8

Depends R (>= 4.1.0)

Imports clue, igraph (>= 2.0.0), rlang

Suggests testthat (>= 3.0.0), ggplot2, gganimate, gifski, ggalluvial, RColorBrewer, dplyr, tidyr, peacesciencer

RoxygenNote 7.3.3

Config/testthat/edition 3

URL https://github.com/jfedgerton/dynamic_multiplex

BugReports https://github.com/jfedgerton/dynamic_multiplex/issues

NeedsCompilation no

Author Jared Edgerton [aut, cre]

Maintainer Jared Edgerton <jared.edgerton@gmail.com>

Repository CRAN

Date/Publication 2026-08-07 16:00:08 UTC

Contents

animate_multilayer_gif 2
bootstrap_multilayer 3

community_est 5
co_assignment_ci 7
extract_meta_membership 8
fit_multilayer_hungarian 9
fit_multilayer_identity_ties 11
fit_multilayer_jaccard 13
fit_multilayer_overlap 14
fit_multilayer_weighted_jaccard 16
fit_multilayer_weighted_overlap 18
plot_multilayer_alluvial 20
plot_multilayer_series 21
simulate_and_fit_multilayer 22

Index **24**

`animate_multilayer_gif` *Animate multiplex networks as a GIF by layer*

Description

Creates one frame per layer and colors nodes by community using colorblind-friendly ColorBrewer palettes (`Set2` or `Dark2`).

Usage

```
animate_multilayer_gif(
  layers,
  fit = NULL,
  community_memberships = NULL,
  output_file,
  directed = FALSE,
  fps = 2,
  width = 800,
  height = 600,
  layout = "layout_with_fr",
  palette = "Dark2"
)
```

Arguments

`layers`	List of `igraph` objects or square adjacency matrices.
`fit`	Optional fit object from `fit_multilayer_jaccard()`, `fit_multilayer_overlap()`, or `fit_multilayer_identity_ties()`.
`community_memberships`	Optional list of membership vectors (one per layer). Ignored when `fit` is provided.

output_file	Output GIF file path. Required; the function writes the animation to this location and nowhere else. Use tempfile(fileext = ".gif") to write to the session temporary directory.
directed	Logical; if TRUE, adjacency matrices are treated as directed.
fps	Frames per second.
width	Width of GIF in pixels.
height	Height of GIF in pixels.
layout	igraph layout function name. Defaults to "layout_with_fr".
palette	ColorBrewer qualitative palette, one of "Set2" or "Dark2".

Value

The path to the created GIF.

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE) &&
    requireNamespace("gganimate", quietly = TRUE) &&
    requireNamespace("gifski", quietly = TRUE)) {
  set.seed(123)
  layers <- lapply(1:3, function(i) {
    m <- matrix(rbinom(64, 1, 0.35), nrow = 8)
    m <- pmax(m, t(m))
    diag(m) <- 0
    m
  })
  fit <- fit_multilayer_jaccard(layers, algorithm = "louvain")
  gif_path <- animate_multilayer_gif(
    layers,
    fit = fit,
    output_file = tempfile(fileext = ".gif"),
    fps = 2,
    width = 200,
    height = 150
  )
}
```

---

bootstrap_multilayer *Bootstrap confidence intervals for multilayer community detection*

Description

Quantifies the uncertainty of multilayer community detection by refitting communities on `n_boot` resampled networks, using a parametric network bootstrap: within- and between-community edge probabilities (and, for weighted networks, edge-weight pools) are estimated from the observed network using the point-estimate partition, and each replicate redraws the full edge set from those estimates. This reproduces the variability of fresh data, including which edges exist.

Uncertainty is quantified on the cross-layer *meta-communities* (the tracked partition from the second-stage detection), not the independently-detected per-layer communities. Co-assignment therefore answers "do these two nodes belong to the same persistent community," and the community count is the number of meta-communities per layer.

Versions before 1.1.0 instead used a Bayesian bootstrap on edge weights (Exponential(1) multipliers on a fixed topology). That scheme was removed: because it never varies which edges exist, it understates the variability of fresh data, and in simulation studies confidence intervals built from it undercovered substantially (pairwise co-assignment intervals covered ~45-48 percent at a nominal 95 percent).

Usage

```
bootstrap_multilayer(
  layers,
 fit_type = c("jaccard", "overlap", "weighted_jaccard", "weighted_overlap", "identity"),
  algorithm = c("louvain", "leiden"),
  n_boot = 100,
  layer_links = NULL,
  min_similarity = 0,
  resolution_parameter = 1,
  directed = FALSE,
  seed = NULL,
  objective = NULL
)
```

Arguments

`layers`	List of adjacency matrices or igraph objects.
`fit_type`	One of `"jaccard"`, `"overlap"`, `"weighted_jaccard"`, `"weighted_overlap"`, `"identity"`.
`algorithm`	Community detection algorithm: `"louvain"` or `"leiden"`.
`n_boot`	Number of bootstrap replicates.
`layer_links`	Optional data.frame defining layer connectivity.
`min_similarity`	Minimum weighted similarity for interlayer ties.
`resolution_parameter`	Resolution parameter for community detection.
`directed`	Logical; if `TRUE`, treat networks as directed.
`seed`	Optional random seed for reproducibility.
`objective`	One of "cpm" or "modularity" for directed networks only

Value

A list of class "multilayer_bootstrap" with components:

n_boot Number of completed bootstrap replicates.

co_assignment Per-layer co-assignment probability matrices on the meta-communities (probability two nodes share a persistent community).

node_stability Per-layer vectors giving the fraction of replicates in which each node was assigned to its modal community.

modularity_samples Per-layer vectors of bootstrap modularity values.

community_count_reproducibility Per-layer numeric vector: the share of completed replicates whose meta-community count equals the observed-network count. A descriptive stability measure. The raw per-replicate community counts are intentionally not returned.

point_estimate The fit result from the original data.

Examples

```
set.seed(123)
layers <- lapply(1:3, function(i) {
  m <- matrix(rbinom(64, 1, 0.35), nrow = 8)
  m <- pmax(m, t(m))
  diag(m) <- 0
  m
})
boot <- bootstrap_multilayer(
  layers,
  fit_type = "jaccard",
  algorithm = "louvain",
  n_boot = 5,
  seed = 123
)
names(boot)
```

---

community_est *Summarize bootstrap community-count reproducibility (meta-communities)*

Description

Reports, for each layer, the community count from the observed network together with its *bootstrap reproducibility*: the proportion of bootstrap replicates in which the fitted number of communities equals the observed-network count. It also returns mean node stability, per-node stability, and the co-assignment matrices.

Usage

```
community_est(boot_result)
```

Arguments

boot_result	Output from bootstrap_multilayer.

Value

A list with components:

community_count Data frame with columns layer, estimate (the observed-network meta-community count), and reproducibility (share of bootstrap replicates whose community count equals estimate, in [0, 1]).

report Character vector, one plain-language sentence per layer.

mean_node_stability Data frame with columns layer, mean_stability.

node_stability Per-layer stability vectors.

co_assignment Per-layer co-assignment matrices.

Why this is not a confidence interval

Earlier versions returned a percentile community_count_ci. It was replaced in version 1.1.0 with a reproducibility summary because the interval's coverage is not robust to model misspecification. In a large simulation study the nominal 95 percent community-count interval covered the truth at or above the nominal level on well-specified planted-partition networks (about 0.99 for n >= 100 nodes), but coverage collapsed to about 0.62 when community sizes were strongly skewed, and no observable diagnostic reliably separated the trustworthy cases from the rest. Rather than ship an interval that silently undercovers, the function now reports how often the community count reproduces under resampling. This is a descriptive stability measure, not a calibrated interval: it makes no claim about the probability that any range contains the true community count. For a validated interval, use co_assignment_ci, whose node-pair coverage held across the same misspecification stress tests. The raw per-replicate community counts are intentionally not exposed anywhere in the package output; only this reproducibility summary is returned.

See Also

co_assignment_ci for calibrated node-pair co-assignment intervals.

Examples

```
set.seed(123)
layers <- lapply(1:3, function(i) {
  m <- matrix(rbinom(64, 1, 0.35), nrow = 8)
  m <- pmax(m, t(m))
  diag(m) <- 0
  m
})
boot <- bootstrap_multilayer(
  layers,
  fit_type = "jaccard",
  algorithm = "louvain",
  n_boot = 5,
  seed = 123
```

```
)
est <- community_est(boot)
est$community_count
est$report
```

`co_assignment_ci` *Wilson confidence intervals for node-pair co-assignment*

Description

For every pair of nodes in every layer, computes a Wilson score interval for the co-clustering propensity: the probability that the fitted community detection procedure places the two nodes in the same community when the data are perturbed. The point estimate is the co-assignment probability from `bootstrap_multilayer` (the share of bootstrap replicates in which the pair was co-assigned), and the interval treats the `n_boot` replicates as binomial draws.

Because co-assignment is label-invariant (it never compares community labels across replicates, only whether two nodes sit together), it avoids the label-switching problem that makes per-node membership intervals ill-defined.

Usage

```
co_assignment_ci(boot_result, alpha = 0.05)
```

Arguments

`boot_result`	Output from `bootstrap_multilayer`.
`alpha`	Significance level (default 0.05 for 95 percent intervals).

Value

A list with one element per layer. Each element is a list with components:

estimate n x n matrix of co-assignment probabilities.

lower n x n matrix of Wilson lower bounds.

upper n x n matrix of Wilson upper bounds.

Diagonals are 1 by construction.

Warning

These intervals quantify the stability of the detection procedure, not the probability that two nodes truly share a community. The Wilson interval is exact for the binomial sampling of bootstrap replicates; its coverage of the fresh-data co-clustering propensity was evaluated in the package's simulation study (see the package NEWS for the release in which validation results were incorporated). Interpret cautiously on networks with fewer than 100 nodes, where community detection itself is unstable.

See Also

community_est for community-count point estimates and node stability summaries.

Examples

```
set.seed(123)
layers <- lapply(1:3, function(i) {
  m <- matrix(rbinom(64, 1, 0.35), nrow = 8)
  m <- pmax(m, t(m))
  diag(m) <- 0
  m
})
boot <- bootstrap_multilayer(
  layers,
  fit_type = "jaccard",
  algorithm = "louvain",
  n_boot = 5,
  seed = 123
)
pci <- co_assignment_ci(boot, alpha = 0.05)
pci[[1]]$estimate[1:4, 1:4]
pci[[1]]$lower[1:4, 1:4]
```

---

extract_meta_membership

Extract cross-layer meta-community membership

Description

Returns the tracked (cross-layer) community assignment produced by the second-stage detection: for each layer, a vector giving every node's meta-community. Unlike the per-layer layer_communities (detected independently, ignoring the coupling), the meta-communities are the partition that reflects the interlayer ties and any custom layer_links. This is the membership that bootstrap_multilayer validates.

Usage

```
extract_meta_membership(fit)
```

Arguments

fit	A fit object from one of the fit_multilayer_* functions.

Value

A list with one integer vector per layer giving each node's meta-community assignment (node order).

Examples

```
set.seed(123)
layers <- lapply(1:3, function(i) {
  m <- matrix(rbinom(64, 1, 0.35), nrow = 8)
  m <- pmax(m, t(m)); diag(m) <- 0; m
})
fit <- fit_multilayer_jaccard(layers, algorithm = "leiden")
extract_meta_membership(fit)
```

---

fit_multilayer_hungarian

Fit multilayer communities with Hungarian snapshot matching

Description

Detects communities independently in each layer and then tracks them across time by matching community labels between consecutive layers with the Hungarian (optimal linear-sum assignment) algorithm applied to the community overlap (contingency) matrix. This is a two-stage *snapshot-and-match* tracker: unlike the coupling-based fits (`fit_multilayer_jaccard`, `fit_multilayer_overlap`, `fit_multilayer_identity_ties`), communities are found per layer and aligned *post hoc*, not jointly optimised on a coupled supra-graph. It provides the common "independent detection + optimal label matching" baseline (e.g. per-layer Leiden matched with the Hungarian algorithm) as a first-class method with the same interface as the other fits.

Usage

```
fit_multilayer_hungarian(
  layers,
  algorithm = c("louvain", "leiden"),
  resolution_parameter = 1,
  directed = FALSE,
  objective = NULL,
  seed = NULL
)
```

Arguments

`layers`	List of `igraph` objects or square adjacency matrices.
`algorithm`	Community algorithm: `"louvain"` or `"leiden"`.
`resolution_parameter`	Leiden resolution parameter.
`directed`	Logical; if `TRUE`, build directed graphs from adjacency matrices (collapsed to undirected for detection, as elsewhere in the package).
`objective`	One of `"cpm"` or `"modularity"` (directed networks only).

`seed` Optional integer seed for reproducible community detection. When supplied, the global RNG state is saved, the RNG is seeded for the duration of the call, and the previous state is restored on exit, so the caller's random number stream (e.g. bootstrap resampling) is unaffected. Defaults to `NULL` (detection inherits the caller's RNG stream, matching previous behavior; call `set.seed()` beforehand for reproducibility).

Details

Matching is sequential: layer t is aligned to layer $t-1$. For each adjacent pair the community overlap matrix is completed to a square cost matrix and `solve_LSAP` maximises total overlap; unmatched current communities receive fresh labels. The result is a consistently-labelled sequence suitable for `extract_meta_membership`, `bootstrap_multilayer`, and the plotting helpers.

Value

A list of class `"multilayer_community_fit"` with components:

layer_communities Per-layer community detection (each with `membership` and `communities`), detected independently.

meta_communities The tracked partition: one integer vector per layer where labels are aligned across consecutive layers by Hungarian matching on community overlap. A community with no positive-overlap match is given a new label (a birth); labels not carried forward are treated as deaths. See `extract_meta_membership`.

interlayer_ties `NULL` - no coupled supra-graph is built.

method `"hungarian"`.

layer_links Sequential adjacent-layer links (the chain the matcher walks).

See Also

`fit_multilayer_jaccard`, `fit_multilayer_overlap`, `fit_multilayer_identity_ties` for coupling-based tracking.

Examples

```
set.seed(123)
layers <- lapply(1:3, function(i) {
  m <- matrix(rbinom(64, 1, 0.35), nrow = 8)
  m <- pmax(m, t(m)); diag(m) <- 0; m
})
fit <- fit_multilayer_hungarian(layers, algorithm = "leiden")
extract_meta_membership(fit)
```

`fit_multilayer_identity_ties` *Fit multilayer communities with identity interlayer ties*

Description

Runs Louvain or Leiden community detection for each layer and creates interlayer ties between the same node in selected adjacent layers. With `allow_unequal_nodes = TRUE`, layers may contain different node sets; only nodes present in both layers of a linked pair receive interlayer edges. Nodes are matched by `V(g)$name` when available, or by vertex index otherwise.

For this specification the cross-layer `meta_communities` are obtained by **Mucha (2010) multislice modularity**: the layers are stacked into a single supra-graph (intra-layer edges are each layer's own adjacency) with interlayer identity edges joining each node to its copies in the coupled layers (weighted by `layer_links`), and one community detection is run on the whole supra-graph. The coupling strength is the layer-link weight.

Usage

```
fit_multilayer_identity_ties(
  layers,
  algorithm = c("louvain", "leiden"),
  layer_links = NULL,
  resolution_parameter = 1,
  omega = 1,
  directed = FALSE,
  objective = NULL,
  seed = NULL,
  allow_unequal_nodes = FALSE
)
```

Arguments

`layers`	List of `igraph` objects or square adjacency matrices.
`algorithm`	Community algorithm: `"louvain"` or `"leiden"`.
`layer_links`	Optional data.frame defining which layers to connect, with columns `from`, `to`, and optional `weight`. If `NULL`, adjacent layers are connected in sequence.
`resolution_parameter`	Leiden resolution parameter. Also forwarded to the multislice supra-graph detection as Mucha's modularity resolution (larger values yield more, smaller meta-communities).
`omega`	Interlayer coupling strength for the multislice supra-graph (Mucha's omega). Multiplies the interlayer identity-edge weights on top of any `layer_links` weights. Larger `omega` couples layers more strongly and, past a point, collapses everything into one meta-community; smaller `omega` decouples toward independent per-layer detection. Use to probe the omega-sensitivity / resolution-limit behavior of multislice modularity.

`directed` Logical; if `TRUE`, build directed graphs from adjacency matrices. Directed layers are collapsed to undirected weighted graphs before community detection (igraph supports Leiden on undirected graphs only; a warning is issued for `algorithm = "leiden"`).

`objective` One of "cpm" or "modularity" for directed networks only

`seed` Optional integer seed for reproducible community detection. When supplied, the global RNG state is saved, the RNG is seeded for the duration of the call, and the previous state is restored on exit, so the caller's random number stream (e.g. bootstrap resampling) is unaffected. Defaults to `NULL` (detection inherits the caller's RNG stream, matching previous behavior; call `set.seed()` beforehand for reproducibility).

`allow_unequal_nodes` Logical; if `TRUE`, layers may contain different node sets (nodes entering or exiting the system), and only nodes present in both layers of a linked pair receive interlayer edges. Defaults to `FALSE`: layers must share the same node universe, as in the other fit functions.

Value

A list of class `"multilayer_community_fit"` with components:

layer_communities Per-layer community detection (each with `membership` and `communities`). Detected independently per layer.

meta_communities The cross-layer tracked partition from the second-stage detection: one integer vector per layer giving each node's meta-community. This is the membership that reflects the interlayer ties and any custom `layer_links`, and the one validated by `bootstrap_multilayer`. See `extract_meta_membership`.

interlayer_ties Interlayer similarity edges between communities (plus self-loops).

layer_links The layer connectivity used.

Directed networks

Directed layers are stored as directed graphs and the interlayer self-loop weighting is directed-aware, but community *detection* collapses directed layers to weighted undirected graphs on both Louvain and Leiden (igraph's detectors are undirected-only). For detection that respects edge direction, use the Python package with `algorithm = "leiden"`.

Examples

```
set.seed(123)
layers <- lapply(1:3, function(i) {
  m <- matrix(rbinom(64, 1, 0.35), nrow = 8)
  m <- pmax(m, t(m))
  diag(m) <- 0
  m
})
fit <- fit_multilayer_identity_ties(layers, algorithm = "louvain")
names(fit)
```

`fit_multilayer_jaccard` *Fit multilayer communities and interlayer weighted Jaccard ties*

Description

Runs Louvain or Leiden community detection for each layer and creates interlayer ties between communities in selected layer pairs using weighted Jaccard similarity.

Usage

```
fit_multilayer_jaccard(
  layers,
  algorithm = c("louvain", "leiden"),
  layer_links = NULL,
  min_similarity = 0,
  resolution_parameter = 1,
  directed = FALSE,
  add_self_loops = TRUE,
  self_loop_multiplier = 1,
  objective = NULL,
  seed = NULL
)
```

Arguments

`layers`	List of `igraph` objects or square adjacency matrices.
`algorithm`	Community algorithm: `"louvain"` or `"leiden"`.
`layer_links`	Optional data.frame defining which layers to connect, with columns `from`, `to`, and optional `weight`. If `NULL`, adjacent layers are connected in sequence.
`min_similarity`	Minimum weighted similarity required to keep an interlayer tie.
`resolution_parameter`	Leiden resolution parameter.
`directed`	Logical; if `TRUE`, build directed graphs from adjacency matrices. Directed layers are collapsed to undirected weighted graphs before community detection (igraph supports Leiden on undirected graphs only; a warning is issued for `algorithm = "leiden"`).
`add_self_loops`	Logical; if `TRUE`, add intra-layer community self-loop ties after similarity computation to mimic aggregation behavior.
`self_loop_multiplier`	Numeric multiplier applied to self-loop weighted ties. The default `1` yields self-loop weighted similarity of `2 * layer_weight` for undirected or `1 * layer_weight` for directed networks, following the Louvain/Leiden community aggregation convention.
`objective`	One of "cpm" or "modularity" for directed networks only

`seed` Optional integer seed for reproducible community detection. When supplied, the global RNG state is saved, the RNG is seeded for the duration of the call, and the previous state is restored on exit, so the caller's random number stream (e.g. bootstrap resampling) is unaffected. Defaults to `NULL` (detection inherits the caller's RNG stream, matching previous behavior; call `set.seed()` beforehand for reproducibility).

Value

A list of class `"multilayer_community_fit"` with components:

layer_communities Per-layer community detection (each with `membership` and `communities`). Detected independently per layer.

meta_communities The cross-layer tracked partition from the second-stage detection: one integer vector per layer giving each node's meta-community. This is the membership that reflects the interlayer ties and any custom `layer_links`, and the one validated by `bootstrap_multilayer`. See `extract_meta_membership`.

interlayer_ties Interlayer similarity edges between communities (plus self-loops).

layer_links The layer connectivity used.

Directed networks

Directed layers are stored as directed graphs and the interlayer self-loop weighting is directed-aware, but community *detection* collapses directed layers to weighted undirected graphs on both Louvain and Leiden (igraph's detectors are undirected-only). For detection that respects edge direction, use the Python package with `algorithm = "leiden"`.

Examples

```
set.seed(123)
layers <- lapply(1:3, function(i) {
  m <- matrix(rbinom(64, 1, 0.35), nrow = 8)
  m <- pmax(m, t(m))
  diag(m) <- 0
  m
})
fit <- fit_multilayer_jaccard(layers, algorithm = "louvain")
names(fit)
fit$layer_communities[[1]]$membership
```

`fit_multilayer_overlap` *Fit multilayer communities and interlayer weighted overlap ties*

Description

Runs Louvain or Leiden community detection for each layer and creates interlayer ties between communities in selected layer pairs using a weighted overlap coefficient.

Usage

```
fit_multilayer_overlap(
  layers,
  algorithm = c("louvain", "leiden"),
  layer_links = NULL,
  min_similarity = 0,
  resolution_parameter = 1,
  directed = FALSE,
  add_self_loops = TRUE,
  self_loop_multiplier = 1,
  objective = NULL,
  seed = NULL
)
```

Arguments

layers	List of igraph objects or square adjacency matrices.
algorithm	Community algorithm: "louvain" or "leiden".
layer_links	Optional data.frame defining which layers to connect, with columns from, to, and optional weight. If NULL, adjacent layers are connected in sequence.
min_similarity	Minimum weighted similarity required to keep an interlayer tie.
resolution_parameter	Leiden resolution parameter.
directed	Logical; if TRUE, build directed graphs from adjacency matrices. Directed layers are collapsed to undirected weighted graphs before community detection (igraph supports Leiden on undirected graphs only; a warning is issued for algorithm = "leiden").
add_self_loops	Logical; if TRUE, add intra-layer community self-loop ties after similarity computation to mimic aggregation behavior.
self_loop_multiplier	Numeric multiplier applied to self-loop weighted ties. The default 1 yields self-loop weighted similarity of 2 * layer_weight for undirected or 1 * layer_weight for directed networks, following the Louvain/Leiden community aggregation convention.
objective	One of "cpm" or "modularity" for directed networks only
seed	Optional integer seed for reproducible community detection. When supplied, the global RNG state is saved, the RNG is seeded for the duration of the call, and the previous state is restored on exit, so the caller's random number stream (e.g. bootstrap resampling) is unaffected. Defaults to NULL (detection inherits the caller's RNG stream, matching previous behavior; call set.seed() beforehand for reproducibility).

Value

A list of class "multilayer_community_fit" with components:

layer_communities Per-layer community detection (each with `membership` and `communities`). Detected independently per layer.

meta_communities The cross-layer tracked partition from the second-stage detection: one integer vector per layer giving each node's meta-community. This is the membership that reflects the interlayer ties and any custom `layer_links`, and the one validated by `bootstrap_multilayer`. See `extract_meta_membership`.

interlayer_ties Interlayer similarity edges between communities (plus self-loops).

layer_links The layer connectivity used.

Directed networks

Directed layers are stored as directed graphs and the interlayer self-loop weighting is directed-aware, but community *detection* collapses directed layers to weighted undirected graphs on both Louvain and Leiden (igraph's detectors are undirected-only). For detection that respects edge direction, use the Python package with `algorithm = "leiden"`.

Examples

```
set.seed(123)
layers <- lapply(1:3, function(i) {
  m <- matrix(rbinom(64, 1, 0.35), nrow = 8)
  m <- pmax(m, t(m))
  diag(m) <- 0
  m
})
fit <- fit_multilayer_overlap(layers, algorithm = "louvain")
names(fit)
```

---

fit_multilayer_weighted_jaccard

Fit multilayer communities and interlayer node-strength weighted Jaccard ties

---

Description

Runs Louvain or Leiden community detection for each layer and creates interlayer ties between communities in selected layer pairs using node-strength weighted Jaccard similarity.

Usage

```
fit_multilayer_weighted_jaccard(
  layers,
  algorithm = c("louvain", "leiden"),
  layer_links = NULL,
  min_similarity = 0,
  resolution_parameter = 1,
```

```
  directed = FALSE,
  add_self_loops = TRUE,
  self_loop_multiplier = 1,
  objective = NULL,
  seed = NULL
)
```

Arguments

layers	List of igraph objects or square adjacency matrices.
algorithm	Community algorithm: "louvain" or "leiden".
layer_links	Optional data.frame defining which layers to connect, with columns from, to, and optional weight. If NULL, adjacent layers are connected in sequence.
min_similarity	Minimum weighted similarity required to keep an interlayer tie.
resolution_parameter	Leiden resolution parameter.
directed	Logical; if TRUE, build directed graphs from adjacency matrices.
add_self_loops	Logical; if TRUE, add intra-layer community self-loop ties.
self_loop_multiplier	Numeric multiplier applied to self-loop weighted ties.
objective	One of "cpm" or "modularity" for directed networks only
seed	Optional integer seed for reproducible community detection. When supplied, the global RNG state is saved, the RNG is seeded for the duration of the call, and the previous state is restored on exit, so the caller's random number stream (e.g. bootstrap resampling) is unaffected. Defaults to NULL (detection inherits the caller's RNG stream, matching previous behavior; call set.seed() beforehand for reproducibility).

Value

A list of class "multilayer_community_fit" with components:

layer_communities Per-layer community detection (each with membership and communities). Detected independently per layer.

meta_communities The cross-layer tracked partition from the second-stage detection: one integer vector per layer giving each node's meta-community. This is the membership that reflects the interlayer ties and any custom layer_links, and the one validated by bootstrap_multilayer. See extract_meta_membership.

interlayer_ties Interlayer similarity edges between communities (plus self-loops).

layer_links The layer connectivity used.

Directed networks

Directed layers are stored as directed graphs and the interlayer self-loop weighting is directed-aware, but community *detection* collapses directed layers to weighted undirected graphs on both Louvain and Leiden (igraph's detectors are undirected-only). For detection that respects edge direction, use the Python package with algorithm = "leiden".

Examples

```
set.seed(123)
layers <- lapply(1:3, function(i) {
  w <- matrix(runif(64), 8, 8) * matrix(rbinom(64, 1, 0.35), 8, 8)
  w <- (w + t(w)) / 2
  diag(w) <- 0
  w
})
fit <- fit_multilayer_weighted_jaccard(layers, algorithm = "louvain")
names(fit)
```

---

fit_multilayer_weighted_overlap

Fit multilayer communities and interlayer node-strength weighted overlap ties

---

Description

Runs Louvain or Leiden community detection for each layer and creates interlayer ties between communities in selected layer pairs using node-strength weighted overlap coefficient.

Usage

```
fit_multilayer_weighted_overlap(
  layers,
  algorithm = c("louvain", "leiden"),
  layer_links = NULL,
  min_similarity = 0,
  resolution_parameter = 1,
  directed = FALSE,
  add_self_loops = TRUE,
  self_loop_multiplier = 1,
  objective = NULL,
  seed = NULL
)
```

Arguments

layers	List of igraph objects or square adjacency matrices.
algorithm	Community algorithm: "louvain" or "leiden".
layer_links	Optional data.frame defining which layers to connect, with columns from, to, and optional weight. If NULL, adjacent layers are connected in sequence.
min_similarity	Minimum weighted similarity required to keep an interlayer tie.
resolution_parameter	Leiden resolution parameter.

`directed` Logical; if `TRUE`, build directed graphs from adjacency matrices.

`add_self_loops` Logical; if `TRUE`, add intra-layer community self-loop ties.

`self_loop_multiplier`
Numeric multiplier applied to self-loop weighted ties.

`objective` One of "cpm" or "modularity" for directed networks only

`seed` Optional integer seed for reproducible community detection. When supplied, the global RNG state is saved, the RNG is seeded for the duration of the call, and the previous state is restored on exit, so the caller's random number stream (e.g. bootstrap resampling) is unaffected. Defaults to `NULL` (detection inherits the caller's RNG stream, matching previous behavior; call `set.seed()` beforehand for reproducibility).

Value

A list of class `"multilayer_community_fit"` with components:

layer_communities Per-layer community detection (each with `membership` and `communities`). Detected independently per layer.

meta_communities The cross-layer tracked partition from the second-stage detection: one integer vector per layer giving each node's meta-community. This is the membership that reflects the interlayer ties and any custom `layer_links`, and the one validated by `bootstrap_multilayer`. See `extract_meta_membership`.

interlayer_ties Interlayer similarity edges between communities (plus self-loops).

layer_links The layer connectivity used.

Directed networks

Directed layers are stored as directed graphs and the interlayer self-loop weighting is directed-aware, but community *detection* collapses directed layers to weighted undirected graphs on both Louvain and Leiden (igraph's detectors are undirected-only). For detection that respects edge direction, use the Python package with `algorithm = "leiden"`.

Examples

```
set.seed(123)
layers <- lapply(1:3, function(i) {
  w <- matrix(runif(64), 8, 8) * matrix(rbinom(64, 1, 0.35), 8, 8)
  w <- (w + t(w)) / 2
  diag(w) <- 0
  w
})
fit <- fit_multilayer_weighted_overlap(layers, algorithm = "louvain")
names(fit)
```

plot_multilayer_alluvial *Plot an alluvial view of community transitions over time*

Description

Plot an alluvial view of community transitions over time

Usage

```
plot_multilayer_alluvial(fit, max_nodes = NULL, palette = "Dark2")
```

Arguments

fit	Fit object from fit_multilayer_jaccard(), fit_multilayer_overlap(), or fit_multilayer_identity_ties().
max_nodes	Optional cap on number of nodes to include for readability.
palette	ColorBrewer qualitative palette, one of "Set2" or "Dark2".

Value

A ggplot object.

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE) &&
    requireNamespace("ggalluvial", quietly = TRUE)) {
  set.seed(123)
  layers <- lapply(1:3, function(i) {
    m <- matrix(rbinom(64, 1, 0.35), nrow = 8)
    m <- pmax(m, t(m))
    diag(m) <- 0
    m
  })
  fit <- fit_multilayer_jaccard(layers, algorithm = "louvain")
  p <- plot_multilayer_alluvial(fit)
}
```

plot_multilayer_series *Plot multiplex networks as wrapped panels*

Description

Draws one panel per layer with nodes colored by community assignment, using a faceted/wrapped layout.

Usage

```
plot_multilayer_series(
  layers,
  fit = NULL,
  community_memberships = NULL,
  directed = FALSE,
  layout = "layout_with_fr",
  ncol = 3,
  palette = "Dark2"
)
```

Arguments

layers	List of igraph objects or square adjacency matrices.
fit	Optional fit object from fit_multilayer_jaccard(), fit_multilayer_overlap(), or fit_multilayer_identity_ties().
community_memberships	Optional list of membership vectors (one per layer). Ignored when fit is provided.
directed	Logical; if TRUE, adjacency matrices are treated as directed.
layout	igraph layout function name. Defaults to "layout_with_fr".
ncol	Number of wrap columns in the panel layout.
palette	ColorBrewer qualitative palette, one of "Set2" or "Dark2".

Value

A ggplot object.

Examples

```
if (requireNamespace("ggplot2", quietly = TRUE)) {
  set.seed(123)
  layers <- lapply(1:3, function(i) {
    m <- matrix(rbinom(64, 1, 0.35), nrow = 8)
    m <- pmax(m, t(m))
    diag(m) <- 0
```

```
    m
  })
  fit <- fit_multilayer_jaccard(layers, algorithm = "louvain")
  p <- plot_multilayer_series(layers, fit = fit)
}
```

`simulate_and_fit_multilayer` *Simulate multiplex layers and fit interlayer models*

Description

Generates synthetic multiplex network layers using a planted partition model, then fits one of the provided interlayer tie strategies.

Usage

```
simulate_and_fit_multilayer(
  n_nodes = 100,
  n_layers = 4,
  n_communities = 4,
  p_in = 0.2,
  p_out = 0.05,
  fit_type = c("jaccard", "overlap", "weighted_jaccard", "weighted_overlap", "identity"),
  algorithm = c("louvain", "leiden"),
  layer_links = NULL,
  min_similarity = 0,
  seed = NULL,
  directed = FALSE
)
```

Arguments

`n_nodes`	Number of nodes per layer.
`n_layers`	Number of temporal layers.
`n_communities`	Number of latent communities.
`p_in`	Probability of an in-community edge.
`p_out`	Probability of an out-community edge.
`fit_type`	One of `"jaccard"`, `"overlap"`, or `"identity"`.
`algorithm`	Community algorithm for fitting: `"louvain"` or `"leiden"`.
`layer_links`	Optional layer connectivity specification.
`min_similarity`	Minimum similarity threshold for overlap-based methods.
`seed`	Optional random seed.
`directed`	Logical; if `TRUE`, simulate directed adjacency matrices.

Value

A list containing simulated layers, true memberships, and fit results.

Examples

```
sim <- simulate_and_fit_multilayer(
  n_nodes = 30,
  n_layers = 3,
  n_communities = 3,
  fit_type = "jaccard",
  algorithm = "louvain",
  seed = 123
)
names(sim)
```

Index

animate_multilayer_gif, 2

bootstrap_multilayer, 3, *6–8*, *10*, *12*, *14*, *16*, *17*, *19*

co_assignment_ci, *6*, 7
community_est, 5, *8*

extract_meta_membership, 8, *10*, *12*, *14*, *16*, *17*, *19*

fit_multilayer_hungarian, 9
fit_multilayer_identity_ties, *9*, *10*, 11
fit_multilayer_jaccard, *9*, *10*, 13
fit_multilayer_overlap, *9*, *10*, 14
fit_multilayer_weighted_jaccard, 16
fit_multilayer_weighted_overlap, 18

plot_multilayer_alluvial, 20
plot_multilayer_series, 21

simulate_and_fit_multilayer, 22
solve_LSAP, *10*