

Package ‘dynamicmultiplex’

October 14, 2026

Title Tracking Communities in Changing Temporal Networks

Version 1.4.0

Description Tools for temporal networks that change over time, where nodes enter and leave, groups form, split, merge and dissolve, and periods differ in size. The temporal fitters track nodes and communities across periods. A community keeps one identifier while it persists, and a newly formed group receives a new one. Two-stage fitters run Louvain or Leiden community detection in each period and link communities in linked periods by Jaccard similarity, the overlap coefficient or node-strength weighted variants. The 'layer_links' argument sets which periods are directly linked, adjacent periods only by default. By default all periods are fitted jointly, so later periods can change earlier labels, and tracking = 'filtered' prevents this. Multislice modularity (Mucha et al. 2010) <[doi:10.1126/science.1184819](https://doi.org/10.1126/science.1184819)> and Hungarian matching are included as comparison methods, and a bootstrap stability score helps choose between the two-stage and 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

Config/testthat/edition 3

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

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

Config/roxygen2/version 8.1.1

NeedsCompilation no

Author Jared Edgerton [aut, cre]

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

Repository CRAN

Date/Publication 2026-10-14 21:42:13 UTC

Contents

animate_multilayer_gif	2
bootstrap_multilayer	4
extract_meta_membership	6
fit_multilayer_hungarian	7
fit_multilayer_identity_ties	9
fit_multilayer_jaccard	11
fit_multilayer_overlap	13
fit_multilayer_weighted_jaccard	16
fit_multilayer_weighted_overlap	18
layers_from_membership	20
partition_stability	22
plot_multilayer_alluvial	23
plot_multilayer_series	24
simulate_and_fit_multilayer	25
Index	27

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.

<code>fit</code>	Optional fit object from any <code>fit_multilayer_*</code> () function. Nodes are colored by the tracked meta_communities, so a color marks the same community in every layer.
<code>community_memberships</code>	Optional list of membership vectors (one per layer). Ignored when <code>fit</code> is provided.
<code>output_file</code>	Output GIF file path. Required; the function writes the animation to this location and nowhere else. Use <code>tempfile(fileext = ".gif")</code> to write to the session temporary directory.
<code>directed</code>	Logical; if TRUE, adjacency matrices are treated as directed.
<code>fps</code>	Frames per second.
<code>width</code>	Width of GIF in pixels.
<code>height</code>	Height of GIF in pixels.
<code>layout</code>	igraph layout function name. Defaults to "layout_with_fr".
<code>palette</code>	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 *Parametric network bootstrap for the stability score*

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.

Since 1.4.0 the package uses the bootstrap only for the stability score s reported by `partition_stability`, which compares fits of the same network. The interval outputs of earlier versions (`co_assignment_ci()`, `community_est()`, the calibrated accuracy floor) were removed because their coverage was not reliable.

Agreement is measured 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("leiden", "louvain"),
  n_boot = 100,
  layer_links = NULL,
  min_similarity = 0,
  resolution_parameter = 1,
  directed = FALSE,
  seed = NULL,
  objective = NULL,
  tracking = "smoothed",
  repair_chains = FALSE,
  repair_min_similarity = 0.2,
  allow_unequal_nodes = FALSE
)
```

Arguments

`layers` List of adjacency matrices or igraph objects.

fit_type	One of "jaccard", "overlap", "weighted_jaccard", "weighted_overlap", "identity".
algorithm	Community detection algorithm: "leiden" (default) or "louvain".
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	Objective for the first-stage (per-layer) detection, passed to the fit function: "modularity" or "cpm" (Leiden only). Defaults to "cpm" for directed layers and "modularity" otherwise.
tracking, repair_chains, repair_min_similarity	Passed to the two-stage fit functions (see fit_multilayer_jaccard) so that every refit uses the same settings as the point estimate. Not available for fit_type = "identity".
allow_unequal_nodes	Logical. If TRUE, layers may hold different node sets (nodes entering and leaving, with or without a change in layer size). Every layer must then carry vertex names; each layer is redrawn on its own nodes and every refit matches nodes by name. Default FALSE: all layers must share one node set.

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.

stability_samples List with matrices nmi and ari (completed replicates x layers): agreement of each replicate's meta-partition with the point-estimate partition. Summarised by [partition_stability](#).

node_jaccard_stability Per-layer vectors: for each node, the mean Jaccard overlap between its replicate community and its point-estimate community.

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)

```

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("leiden", "louvain"),
  resolution_parameter = 1,
  directed = FALSE,
  objective = NULL,
  seed = 123,
  allow_unequal_nodes = FALSE
)

```

Arguments

layers	List of igraph objects or square adjacency matrices.
algorithm	Community algorithm: "leiden" (default) or "louvain".
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	Objective for the per-layer detection: "modularity" or "cpm" (Leiden only). Defaults to "cpm" for directed layers and "modularity" otherwise.
seed	Integer seed for reproducible community detection (default 123, as in the Python package; pass NULL for unseeded 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. Layers must then carry vertex names (or row names on the adjacency matrices); community overlap is counted on the nodes present in both layers of each consecutive pair. Defaults to FALSE.

Details

Matching is sequential: layer t is aligned to layer $t - 1$. Nodes are matched across layers by vertex name when names are present (by position otherwise), so the order in which vertices are stored does not affect the result. 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 et al. (2010) multislice modularity**: each layer keeps its own configuration null model ($k_{is}k_{js}/2m_s$), interlayer identity ties (each node to its copies in the coupled layers, weight = layer-link weight times omega) carry no null term, and the objective is optimised with a generalized Louvain (local moving + aggregation, random restarts). Versions before 1.2.1 ran plain single-graph modularity on the stacked supra-graph, which is a different objective and returns each layer as one community at $\omega \leq 1$.

Usage

```
fit_multilayer_identity_ties(
  layers,
  algorithm = c("leiden", "louvain"),
  layer_links = NULL,
  resolution_parameter = 1,
  omega = 1,
  directed = FALSE,
  objective = NULL,
  seed = 123,
  allow_unequal_nodes = FALSE
)
```

Arguments

layers	List of igraph objects or square adjacency matrices.
algorithm	Community algorithm for the per-layer (first-stage) partitions: "leiden" (default) or "louvain". The multislice meta-communities always use the generalized Louvain optimiser.
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, and every layer must be directed. The first-stage per-layer detection collapses directed layers to undirected weighted graphs (igraph supports Leiden on undirected graphs only; a warning is issued for algorithm = "leiden"). The multislice stage (meta_communities) keeps arc direction: each slice uses the directed null model $k_i^{out} k_j^{in} / m_s$, and the interlayer identity ties stay undirected.
objective	Objective for the first-stage (per-layer) detection stored in layer_communities: "modularity" or "cpm" (CPM requires algorithm = "leiden"). Defaults to "cpm" for directed layers and "modularity" otherwise. It does not affect meta_communities, which always optimize multislice modularity.
seed	Integer seed for reproducible community detection (default 123, as in the Python package; pass NULL for unseeded 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

With directed = TRUE the multislice stage that produces meta_communities respects edge direction (Leicht and Newman's directed null model in each slice). A directed network in which every arc is reciprocated gives the same meta_communities as its undirected twin at the same omega; before version 1.3.2 such a network was fitted as if omega were halved. The first-stage layer_communities still collapse directed layers to weighted undirected graphs (igraph's detectors are undirected-only).

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("leiden", "louvain"),
  layer_links = NULL,
  min_similarity = 0,
  resolution_parameter = 1,
  directed = FALSE,
  add_self_loops = TRUE,
  self_loop_multiplier = 1,
  objective = NULL,
  seed = 123,
  allow_unequal_nodes = FALSE,
  tracking = "smoothed",
  repair_chains = FALSE,
  repair_min_similarity = 0.2
)

```

Arguments

layers	List of igraph objects or square adjacency matrices.
algorithm	Community algorithm: "leiden" (default) or "louvain".
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 $2 * \text{layer_weight}$ for undirected or $1 * \text{layer_weight}$ for directed networks, following the Louvain/Leiden community aggregation convention.
objective	Objective for the first-stage (per-layer) detection: "modularity" or "cpm" (CPM requires algorithm = "leiden"). Defaults to "cpm" for directed layers and "modularity" otherwise. It does not affect the second stage, which always maximizes modularity on the community graph.
seed	Integer seed for reproducible community detection (default 123, as in the Python package; pass NULL for unseeded 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). Layers must then carry vertex names (or row names on the adjacency matrices) so nodes can be matched across layers; only nodes present in both layers of a linked pair contribute to the interlayer similarity. Defaults to FALSE: layers must share the same node universe.
tracking	"smoothed" (default, the 1.3.2 behaviour): stage 2 is fitted on all layers at once, so a layer's meta ids can depend on later layers. "filtered": stage 2 runs forward in time and the meta ids of layers 1..t depend only on layers 1..t (no future information). Filtered mode assumes the layers are in time order and needs a non-NULL seed.
repair_chains	Logical. If TRUE, rejoin meta-communities that stage 2 cut along time: a group that ends at layer s and one that starts at a linked layer t are joined when their communities are mutual best matches with similarity at least repair_min_similarity. Default FALSE (1.3.2 behaviour).
repair_min_similarity	Similarity floor for chain repair, between 0 and 1. Default 0.2.

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("leiden", "louvain"),
  layer_links = NULL,
  min_similarity = 0,
  resolution_parameter = 1,
  directed = FALSE,
  add_self_loops = TRUE,
  self_loop_multiplier = 1,
```

```

    objective = NULL,
    seed = 123,
    allow_unequal_nodes = FALSE,
    tracking = "smoothed",
    repair_chains = FALSE,
    repair_min_similarity = 0.2
)

```

Arguments

layers	List of igraph objects or square adjacency matrices.
algorithm	Community algorithm: "leiden" (default) or "louvain".
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 $2 * \text{layer_weight}$ for undirected or $1 * \text{layer_weight}$ for directed networks, following the Louvain/Leiden community aggregation convention.
objective	Objective for the first-stage (per-layer) detection: "modularity" or "cpm" (CPM requires algorithm = "leiden"). Defaults to "cpm" for directed layers and "modularity" otherwise. It does not affect the second stage, which always maximizes modularity on the community graph.
seed	Integer seed for reproducible community detection (default 123, as in the Python package; pass NULL for unseeded 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 <code>set.seed()</code> beforehand for reproducibility).
allow_unequal_nodes	Logical; if TRUE, layers may contain different node sets (nodes entering or exiting the system). Layers must then carry vertex names (or row names on the adjacency matrices) so nodes can be matched across layers; only nodes present in both layers of a linked pair contribute to the interlayer similarity. Defaults to FALSE: layers must share the same node universe.

tracking	"smoothed" (default, the 1.3.2 behaviour): stage 2 is fitted on all layers at once, so a layer's meta ids can depend on later layers. "filtered": stage 2 runs forward in time and the meta ids of layers 1..t depend only on layers 1..t (no future information). Filtered mode assumes the layers are in time order and needs a non-NULL seed.
repair_chains	Logical. If TRUE, rejoin meta-communities that stage 2 cut along time: a group that ends at layer s and one that starts at a linked layer t are joined when their communities are mutual best matches with similarity at least repair_min_similarity. Default FALSE (1.3.2 behaviour).
repair_min_similarity	Similarity floor for chain repair, between 0 and 1. Default 0.2.

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("leiden", "louvain"),
  layer_links = NULL,
  min_similarity = 0,
  resolution_parameter = 1,
  directed = FALSE,
  add_self_loops = TRUE,
  self_loop_multiplier = 1,
  objective = NULL,
  seed = 123,
  allow_unequal_nodes = FALSE,
  tracking = "smoothed",
  repair_chains = FALSE,
  repair_min_similarity = 0.2
)
```

Arguments

layers	List of igraph objects or square adjacency matrices.
algorithm	Community algorithm: "leiden" (default) or "louvain".
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	Objective for the first-stage (per-layer) detection: "modularity" or "cpm" (CPM requires algorithm = "leiden"). Defaults to "cpm" for directed layers and "modularity" otherwise. It does not affect the second stage, which always maximizes modularity on the community graph.

seed	Integer seed for reproducible community detection (default 123, as in the Python package; pass NULL for unseeded 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 <code>set.seed()</code> beforehand for reproducibility).
allow_unequal_nodes	Logical; if TRUE, layers may contain different node sets (nodes entering or exiting the system). Layers must then carry vertex names (or row names on the adjacency matrices) so nodes can be matched across layers; only nodes present in both layers of a linked pair contribute to the interlayer similarity. Defaults to FALSE: layers must share the same node universe.
tracking	"smoothed" (default, the 1.3.2 behaviour): stage 2 is fitted on all layers at once, so a layer's meta ids can depend on later layers. "filtered": stage 2 runs forward in time and the meta ids of layers 1..t depend only on layers 1..t (no future information). Filtered mode assumes the layers are in time order and needs a non-NULL seed.
repair_chains	Logical. If TRUE, rejoin meta-communities that stage 2 cut along time: a group that ends at layer s and one that starts at a linked layer t are joined when their communities are mutual best matches with similarity at least <code>repair_min_similarity</code> . Default FALSE (1.3.2 behaviour).
repair_min_similarity	Similarity floor for chain repair, between 0 and 1. Default 0.2.

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("leiden", "louvain"),
  layer_links = NULL,
  min_similarity = 0,
  resolution_parameter = 1,
  directed = FALSE,
  add_self_loops = TRUE,
  self_loop_multiplier = 1,
  objective = NULL,
  seed = 123,
  allow_unequal_nodes = FALSE,
  tracking = "smoothed",
  repair_chains = FALSE,
  repair_min_similarity = 0.2
)

```

Arguments

layers	List of igraph objects or square adjacency matrices.
algorithm	Community algorithm: "leiden" (default) or "louvain".
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.

<code>min_similarity</code>	Minimum weighted similarity required to keep an interlayer tie.
<code>resolution_parameter</code>	Leiden resolution parameter.
<code>directed</code>	Logical; if TRUE, build directed graphs from adjacency matrices.
<code>add_self_loops</code>	Logical; if TRUE, add intra-layer community self-loop ties.
<code>self_loop_multiplier</code>	Numeric multiplier applied to self-loop weighted ties.
<code>objective</code>	Objective for the first-stage (per-layer) detection: "modularity" or "cpm" (CPM requires <code>algorithm = "leiden"</code>). Defaults to "cpm" for directed layers and "modularity" otherwise. It does not affect the second stage, which always maximizes modularity on the community graph.
<code>seed</code>	Integer seed for reproducible community detection (default 123, as in the Python package; pass NULL for unseeded 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 <code>set.seed()</code> beforehand for reproducibility).
<code>allow_unequal_nodes</code>	Logical; if TRUE, layers may contain different node sets (nodes entering or exiting the system). Layers must then carry vertex names (or row names on the adjacency matrices) so nodes can be matched across layers; only nodes present in both layers of a linked pair contribute to the interlayer similarity. Defaults to FALSE: layers must share the same node universe.
<code>tracking</code>	"smoothed" (default, the 1.3.2 behaviour): stage 2 is fitted on all layers at once, so a layer's meta ids can depend on later layers. "filtered": stage 2 runs forward in time and the meta ids of layers 1..t depend only on layers 1..t (no future information). Filtered mode assumes the layers are in time order and needs a non-NULL seed.
<code>repair_chains</code>	Logical. If TRUE, rejoin meta-communities that stage 2 cut along time: a group that ends at layer <i>s</i> and one that starts at a linked layer <i>t</i> are joined when their communities are mutual best matches with similarity at least <code>repair_min_similarity</code> . Default FALSE (1.3.2 behaviour).
<code>repair_min_similarity</code>	Similarity floor for chain repair, between 0 and 1. Default 0.2.

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)
```

layers_from_membership

Build one weighted layer per time slice from group memberships

Description

Membership panels (states in IGOs, legislators in caucuses) are usually projected to a node-by-node network whose tie weight counts shared groups. `layers_from_membership()` builds that projection, or one of two size-aware alternatives, slice by slice, and returns layers that the `fit_multilayer_*` functions accept.

Let B be the $N \times M$ 0/1 membership matrix of a slice and s_j the size of group j .

"count" $W = BB^\top$ (number of shared groups), diagonal 0.

"size" $W_{ik} = \sum_j B_{ij}B_{kj}/(s_j - 1)$, so each group adds total weight proportional to its size rather than its size squared.

"kumar" Kumar et al.'s hypergraph reweighting (as in hypernetx 2.4.3). Group weights w_j start at 1; the slice is projected as in "size" with weights w_j and clustered with `algorithm`; each group gets the target $r_j = [\sum_c 1/(1 + n_{jc})](s_j + C)/M$ and $w_j \leftarrow (w_j + r_j)/2$, until $\max_j |w_j - r_j|/2 \leq \text{tol}$. The projection with the final w_j is returned (the reference returns only the partition). The clustering calls the same per-layer routine as the fitters, so `algorithm` and `resolution_parameter` must be the ones you pass to the fitter.

Every slice is built on its own and reseeded with seed: the weights of slice t never depend on any other slice, so no later information enters an earlier layer.

No weighting is safe in general. In small Python simulations (hypotheses, not established results): the count projection failed badly when a few very large groups were present, where size-weighted and Kumar both recovered the planted blocs; Kumar beat size-weighting only under noise groups of the same size as real ones; under small groups bridging blocs, count was best and both size-aware weightings failed; and per-layer Louvain reproduced Kumar's own partition less reliably than Leiden.

Usage

```
layers_from_membership(
    memberships,
    weighting,
    algorithm = NULL,
    resolution_parameter = 1,
    seed = 123,
    max_iter = 50L,
    tol = 0.01,
    min_group_size = 2L,
    rescale = "none"
)
```

Arguments

memberships	List with one entry per slice: a 0/1 matrix with node labels as row names (columns are groups), or a data frame with exactly the columns node and group.
weighting	"count", "size" or "kumar". Required; there is no default.
algorithm	"leiden" or "louvain"; required for "kumar" and not accepted otherwise.
resolution_parameter	Resolution for Kumar's clustering (> 0).
seed	Seed for Kumar's clustering, reused for every slice (default 123). The caller's RNG state is restored on exit.
max_iter	Maximum number of Kumar reweighting passes (default 50). Failing to converge is an error.
tol	Convergence tolerance on the largest half-change in a group weight.
min_group_size	Groups with fewer members (default 2) are dropped, with a warning.
rescale	"none" (default) or "mean" (each slice divided by its mean nonzero tie). Matters only where tie scale matters, such as the multislice coupling ω ; modularity-based two-stage fits are scale-invariant.

Details

Projected ties are not independent across pairs (one group creates many ties), so the bootstrap stability score has not been checked for layers built this way.

Value

A list of weighted undirected igraph graphs, one per slice, with vertex names. Pass `allow_unequal_nodes = TRUE` to the fitter when node sets differ across slices. The graph attribute `dm_membership` holds weighting, `group_weights` (final w_j by group), `iterations`, `final_diff`, `groups_dropped`, `rescale` and `partition` (Kumar's final partition, a named integer vector; NULL otherwise).

Examples

```
m1 <- data.frame(node = c("a", "b", "c", "c", "d"), group = c("g1", "g1", "g1", "g2", "g2"))
m2 <- data.frame(node = c("a", "b", "b", "d"), group = c("g1", "g1", "g3", "g3"))
layers <- layers_from_membership(list(m1, m2), weighting = "size")
igraph::E(layers[[1]])$weight
```

partition_stability	<i>Bootstrap stability score of a tracked partition</i>
---------------------	---

Description

Summarises a `bootstrap_multilayer` result into the stability score s : the mean agreement (NMI by default, or ARI) between each bootstrap replicate's meta-partition and the point-estimate meta-partition, averaged over layers and replicates. s is a point estimate with Monte Carlo standard error `stability_mc_se`. It has no validated mapping to accuracy: use it only to compare two fits of the same network (for example DynMux against multislice), never as an estimate of how accurate either fit is. Since 1.4.0 the package reports no interval or accuracy floor.

Usage

```
partition_stability(boot_result, metric = c("nmi", "ari"))
```

Arguments

<code>boot_result</code>	Output from <code>bootstrap_multilayer</code> (version 1.3.0 or later, which records <code>stability_samples</code>).
<code>metric</code>	"nmi" (default) or "ari".

Value

A list of class "partition_stability":

stability Stability score s (mean over replicates and layers).

stability_by_layer Per-layer mean agreement.

stability_mc_se Monte Carlo standard error of s from the replicate spread.

metric The agreement measure used.

report One-sentence plain-language summary.

See Also[bootstrap_multilayer](#)**Examples**

```

set.seed(123)
layers <- lapply(1:3, function(i) {
  m <- matrix(rbinom(400, 1, 0.3), nrow = 20)
  m <- pmax(m, t(m)); diag(m) <- 0; m
})
boot <- bootstrap_multilayer(layers, fit_type = "jaccard",
                             algorithm = "leiden", n_boot = 10, seed = 123)
ps <- partition_stability(boot)
ps$stability; ps$report

```

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 any fit_multilayer_*() function. Flows follow each actor (by vertex name when the layers are named) across the tracked meta_communities.
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 any fit_multilayer_*() function. Nodes are colored by the tracked meta_communities, so a color marks the same community in every layer.
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("leiden", "louvain"),
  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.

<code>fit_type</code>	One of "jaccard", "overlap", "weighted_jaccard", "weighted_overlap", or "identity".
<code>algorithm</code>	Community algorithm for fitting: "leiden" (default) or "louvain".
<code>layer_links</code>	Optional layer connectivity specification.
<code>min_similarity</code>	Minimum similarity threshold for overlap-based methods.
<code>seed</code>	Optional random seed.
<code>directed</code>	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`, [4](#), [6](#), [8](#), [10](#), [13](#), [15](#),
[17](#), [19](#), [22](#), [23](#)

`extract_meta_membership`, [6](#), [8](#), [10](#), [13](#), [15](#),
[17](#), [19](#)

`fit_multilayer_hungarian`, [7](#)

`fit_multilayer_identity_ties`, [7](#), [8](#), [9](#)

`fit_multilayer_jaccard`, [5](#), [7](#), [8](#), [11](#)

`fit_multilayer_overlap`, [7](#), [8](#), [13](#)

`fit_multilayer_weighted_jaccard`, [16](#)

`fit_multilayer_weighted_overlap`, [18](#)

`layers_from_membership`, [20](#)

`partition_stability`, [4](#), [5](#), [22](#)

`plot_multilayer_alluvial`, [23](#)

`plot_multilayer_series`, [24](#)

`simulate_and_fit_multilayer`, [25](#)

`solve_LSAP`, [8](#)