Package: networkflow (via r-universe)

November 29, 2026

Title A Research Pipeline for Projected and Dynamic Networks

Version 0.1.0.9000

Description Organises the study of networks built from tabular research data (articles and the references they cite, authors and their papers, accounts and what they share) as a sequence of explicit steps. It projects two-mode data (bibliographic coupling, co-citation, coupling of authors, co-authorship...) onto one-mode networks, weighted by bibliometric similarity measures or filtered by statistical backbones, for a whole corpus or one network per time window; detects communities with the 'igraph' algorithms (Leiden, Louvain, Infomap...); links the communities of successive windows into dynamic clusters; names them with tf-idf, central nodes or, optionally, a language model reached through 'ellmer' (a local model or an online service); characterises the roles of nodes; and draws networks and alluvial plots. Networks are 'tidygraph' objects.

License MIT + file LICENSE

URL https://github.com/agoutsmedt/networkflow, https://agoutsmedt.github.io/networkflow/

BugReports https://github.com/agoutsmedt/networkflow/issues

Depends R (>= 4.1.0)

Imports biblionetwork (>= 0.2.0), cli, data.table (>= 1.15.0), dplyr, ggplot2, ggraph, ggrepel, grDevices, igraph (>= 2.0.0), magrittr, Rdpack, rlang (>= 1.1.0), stats, tidygraph, tidyr, utils, withr

Suggests backbone, DBI, duckdb, ellmer (>= 0.4.1), ggalluvial, lifecycle, scico, testthat (>= 3.0.0), textstem, tidytext, uwot

RdMacros Rdpack

Config/testthat/edition 3

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

Config/roxygen2/version 8.1.1

Config/pak/sysreqs libfontconfig1-dev libfreetype6-dev libfribidi-dev libglpk-dev libharfbuzz-dev libicu-dev libjpeg-dev libpng-dev libxml2-dev

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

Date/Publication 2026-11-29 17:58:23 UTC

RemoteUrl https://github.com/agoutsmedt/networkflow

RemoteRef HEAD

RemoteSha d31f127975cb92655f3252afa7e7cc9df63cef41

Contents

add_clusters . . . 3
add_node_roles . . . 6
Authors_stagflation . . . 10
build_dynamic_networks . . . 10
build_network . . . 18
color_networks . . . 21
Edges_coupling . . . 22
extract_tfidf . . . 23
filter_components . . . 26
layout_networks . . . 27
merge_diagnostics . . . 30
merge_dynamic_clusters . . . 32
minimize_crossing_alluvial . . . 35
name_clusters . . . 38
networks_to_alluv . . . 43
Nodes_coupling . . . 45
Nodes_stagflation . . . 46
plot_alluvial . . . 47
plot_merge_diagnostics . . . 49
plot_networks . . . 51
prepare_label_networks . . . 53
Ref_stagflation . . . 54

Index **56**

`add_clusters` *Detect communities in networks*

Description

[Experimental]

The second step of the pipeline. In a bibliographic coupling network, groups of articles that cite the same literature, much more among themselves than with the rest of the network, are a usual operational definition of a research specialty or a research front (Boyack and Klavans 2010; Morris and Van der Veer Martens 2008). `add_clusters()` finds such groups (communities, or clusters) with one of five algorithms of `igraph`, adds the cluster of each node and of each edge, and the relative size of each cluster. It works on one `tbl_graph` or on each graph of a list, as produced by `build_dynamic_networks()`.

Most methods search for the partition that maximises the *modularity* (Newman and Girvan 2004): the share of the edge weight that falls within clusters, minus the share expected if edges were placed at random given the weighted degree k_i of each node,

$$Q = \frac{1}{2m} \sum_{i,j} \left(A_{ij} - \gamma \frac{k_i k_j}{2m} \right) \delta(c_i, c_j),$$

with A_{ij} the weight of the edge between i and j, m the total weight, c_i the cluster of i, and γ the resolution (1 in the original definition).

Usage

```
add_clusters(
  graphs,
  weights = NULL,
 clustering_method = c("leiden", "louvain", "fast_greedy", "infomap", "walktrap"),
  objective_function = c("modularity", "CPM"),
  resolution = 1,
  n_iterations = -1,
  n_groups = NULL,
  node_weights = NULL,
  trials = 10,
  steps = 4,
  verbose = TRUE,
  seed = NA
)
```

Arguments

`graphs`	A `tbl_graph` or a list of `tbl_graphs`.
`weights`	Edge weights. `NULL` (default) uses the `weight` edge column when it exists, `NA` ignores it, and a numeric vector gives the weights directly. Weights are read as strengths of ties: a larger weight pulls two nodes into the same cluster.

`clustering_method`
: The algorithm, to choose explicitly among `"leiden"`, `"louvain"`, `"fast_greedy"`, `"infomap"` and `"walktrap"` (see the details).

`objective_function`
: For Leiden, the quality function to maximise: `"modularity"` or `"CPM"` (constant Potts model). Required for Leiden.

`resolution`
: For Leiden, the resolution γ. Higher values give more and smaller clusters.

`n_iterations`
: For Leiden, the number of iterations of the algorithm. The default, `-1`, iterates until the partition no longer improves (see `igraph::cluster_leiden()`). A fixed number is much slower on large networks for no gain: on a window of 8,700 articles and 116,000 edges, 1,000 iterations took 44 s, against 0.6 s for `-1`, for the same modularity.

`n_groups`
: For fast greedy and walktrap, an optional number of clusters at which to cut the dendrogram.

`node_weights`
: For Leiden and Infomap, optional name of a node column with node weights (see `igraph::cluster_leiden()` and `igraph::cluster_infomap()`).

`trials`
: For Infomap, the number of attempts to partition the network.

`steps`
: For walktrap, the length of the random walks.

`verbose`
: Logical. If `TRUE` (default), reports the number of clusters and the size of the largest one.

`seed`
: Optional integer. Leiden, Louvain and Infomap are stochastic: two runs can give different partitions. A seed makes the result reproducible. It is set locally (with `withr::local_seed()`): the random-number stream of the session is left unchanged.

Details

The algorithms.

- `"leiden"` (Traag et al. 2019), the default choice of recent bibliometric studies, improves on Louvain: it guarantees that clusters are connected, and iterating it refines the partition. It can maximise the modularity or the constant Potts model (CPM) (Traag et al. 2011), which compares the internal weight e_c of each cluster with γ times its number of pairs,

$$H = \sum_c \left(e_c - \gamma \binom{n_c}{2} \right) :$$

  a cluster is worth creating when its density exceeds γ. Unlike the modularity, CPM does not depend on the size of the network, which makes clusters more comparable across networks of different sizes, but γ must then be set on the scale of the edge weights (with weights well below 1, `resolution` = 1 yields almost only singletons).
- `"louvain"` (Blondel et al. 2008), the greedy multi-level optimisation of modularity that Leiden refines.
- `"fast_greedy"` (Clauset et al. 2004), an agglomerative optimisation of modularity, deterministic.
- `"walktrap"` (Pons and Latapy 2006), which groups nodes that short random walks tend to visit together.

- `"infomap"` (Rosvall and Bergstrom 2008), which finds the partition that best compresses the description of a random walk on the network.

All algorithms are called from `igraph` (`igraph::cluster_leiden()`, `igraph::cluster_louvain()`, `igraph::cluster_fast_greedy()`, `igraph::cluster_walktrap()`, `igraph::cluster_infomap()`), and all use the edge weights unless `weights = NA`.

The resolution. Modularity has a resolution limit: in large networks, it can merge small but clear-cut communities (Fortunato 2010). There is no single right number of clusters: a network of a discipline can be described at the level of fields or of specialties (Sjögårde and Ahlgren 2020). Changing `resolution`, and checking that the main clusters persist across close values and across seeds (Traag et al. 2013), is part of the analysis.

The output columns. With `clustering_method = "leiden"`, nodes get `cluster_leiden` (identifiers numbered by decreasing cluster size, "01" being the largest, written with two digits so that they sort correctly) and `size_cluster_leiden` (the share of the nodes of the graph in the cluster). Edges get `cluster_leiden_from`, `cluster_leiden_to` and `cluster_leiden`, the latter being the common cluster of both ends, or "00" for edges between clusters. The edge cluster is what lets `color_networks()` give within-cluster edges the colour of their cluster. Other methods produce the same columns with their own suffix.

Value

The same object as `graphs` (a `tbl_graph` or a list of `tbl_graphs`) with the cluster columns described in the details, with the node table active.

References

Blondel VD, Guillaume J, Lambiotte R, Lefebvre E (2008). "Fast Unfolding of Communities in Large Networks." *Journal of Statistical Mechanics: Theory and Experiment*, **2008**(10), P10008. doi:10.1088/17425468/2008/10/P10008.

Boyack KW, Klavans R (2010). "Co-Citation Analysis, Bibliographic Coupling, and Direct Citation: Which Citation Approach Represents the Research Front Most Accurately?" *Journal of the American Society for Information Science and Technology*, **61**(12), 2389–2404. doi:10.1002/asi.21419.

Clauset A, Newman MEJ, Moore C (2004). "Finding Community Structure in Very Large Networks." *Physical Review E*, **70**(6), 066111. doi:10.1103/PhysRevE.70.066111.

Fortunato S (2010). "Community Detection in Graphs." *Physics Reports*, **486**(3), 75–174. doi:10.1016/j.physrep.2009.11.002.

Morris SA, Van der Veer Martens B (2008). "Mapping Research Specialties." *Annual Review of Information Science and Technology*, **42**(1), 213–295. doi:10.1002/aris.2008.1440420113.

Newman MEJ, Girvan M (2004). "Finding and Evaluating Community Structure in Networks." *Physical Review E*, **69**(2), 026113. doi:10.1103/PhysRevE.69.026113.

Pons P, Latapy M (2006). "Computing Communities in Large Networks Using Random Walks." *Journal of Graph Algorithms and Applications*, **10**(2), 191–218. doi:10.7155/jgaa.00124.

Rosvall M, Bergstrom CT (2008). "Maps of Random Walks on Complex Networks Reveal Community Structure." *Proceedings of the National Academy of Sciences*, **105**(4), 1118–1123. doi:10.1073/pnas.0706851105.

Sjögårde P, Ahlgren P (2020). "Granularity of Algorithmically Constructed Publication-Level Classifications of Research Publications: Identification of Specialties." *Quantitative Science Studies*, **1**(1), 207–238. doi:10.1162/qss_a_00004.

Traag VA, Krings G, Van Dooren P (2013). "Significant Scales in Community Structure." *Scientific Reports*, **3**, 2930. doi:10.1038/srep02930.

Traag VA, Van Dooren P, Nesterov Y (2011). "Narrow Scope for Resolution-Limit-Free Community Detection." *Physical Review E*, **84**(1), 016114. doi:10.1103/PhysRevE.84.016114.

Traag VA, Waltman L, van Eck NJ (2019). "From Louvain to Leiden: Guaranteeing Well-Connected Communities." *Scientific Reports*, **9**(1), 5233. doi:10.1038/s4159801941695z.

See Also

`merge_dynamic_clusters()` to follow clusters across time windows, `name_clusters()` to label them, `add_node_roles()` to characterise the position of nodes within and between clusters.

Examples

```
network <- build_network(
  nodes = subset(Nodes_stagflation, source_type == "Stagflation"),
  directed_edges = Ref_stagflation,
  source_id = "source_id",
  target_id = "target_id",
  method = "similarity",
  edges_threshold = 1
)

network <- add_clusters(
  network,
  clustering_method = "leiden",
  objective_function = "modularity",
  resolution = 1,
  seed = 123
)
table(tidygraph::as_tibble(network, active = "nodes")$cluster_leiden)
```

`add_node_roles` *Characterise the role of nodes within and between clusters*

Description

[Experimental]

Two nodes of the same cluster can play very different roles: one may be tied to almost every member of its cluster, another may sit at the border and connect several clusters. `add_node_roles()` computes the two measures of the "functional cartography" of Guimerà and Nunes Amaral (2005) and classifies each node into one of their seven roles.

With κ_i the weight of the edges of node i towards its own cluster s_i, the **within-module degree** z-score measures how strongly i is connected to its cluster compared with the other members:

$$z_i = \frac{\kappa_i - \bar{\kappa}_{s_i}}{\sigma_{\kappa_{s_i}}}.$$

With k_{is} the weight of the edges of i towards cluster s and k_i its total weight, the **participation coefficient** is close to 0 when all the ties of i stay within one cluster, and close to 1 when they are spread evenly across clusters:

$$P_i = 1 - \sum_s \left(\frac{k_{is}}{k_i}\right)^2.$$

Usage

```
add_node_roles(
  graphs,
  cluster_id = "dynamic_cluster_leiden",
  weights = NULL,
  z_threshold = 2.5
)
```

Arguments

`graphs`	A `tbl_graph` or a list of `tbl_graph`s with a cluster column.
`cluster_id`	Character scalar, the node column with the clusters. The default, `"dynamic_cluster_leiden"`, fits the output of `merge_dynamic_clusters()`; use `"cluster_leiden"` after `add_clusters()` alone.
`weights`	Edge weights, as in `add_clusters()` and the functions of `tidygraph`. `NULL` (default) uses the `weight` edge column if there is one, and treats the graph as unweighted otherwise. A character scalar names another edge column. `NA` ignores the weights.
`z_threshold`	Number, the within-module z-score above which a node is a hub (2.5, the value of Guimerà and Amaral).

Details

The roles combine the two measures, with the thresholds of Guimerà and Nunes Amaral (2005):

Role	z_i	P_i
R1 ultra-peripheral	below `z_threshold`	below 0.05
R2 peripheral	below `z_threshold`	0.05 to 0.62

R3 non-hub connector	below `z_threshold`	0.62 to 0.80
R4 non-hub kinless	below `z_threshold`	above 0.80
R5 provincial hub	at least `z_threshold`	below 0.30
R6 connector hub	at least `z_threshold`	0.30 to 0.75
R7 kinless hub	at least `z_threshold`	above 0.75

Reading the roles. The two measures answer two questions: *how central is this node in its own cluster?* (z_i) and *where do its ties go?* (P_i). What a role means depends on what the nodes and the ties are. For the stagflation data of the package (documents linked by bibliographic coupling, Leiden clusters, resolution 1), the examples are:

- *Provincial hub* (R5): the central document of a literature, tied to most of its members and to almost nothing else. In a coupling network, it shares references with most documents of its cluster: Lansing's "Exploring the causes of the great inflation" (2000) in the cluster on the great inflation, Nordhaus's "Who's afraid of a big bad oil shock?" (2007) in the cluster on oil. In a co-citation network, the same role is played by the classics that are cited with most works of a literature.
- *Connector hub* (R6): central in its cluster and tied to other clusters: a document that is embedded in a literature while drawing on others. Pindyck's "Energy price increases and macroeconomic policy" (1980) is the only one: central in the cluster on productivity, oil and supply, it shares references with several other clusters.
- *Kinless hub* (R7): central in its cluster, with ties spread evenly over all clusters. It requires a cluster that is itself not very separate from the others; there is none in the stagflation example.
- *Non-hub connector* (R3): a document whose ties are spread over several clusters without being central in any: a bridge. Goldstein's "Downward price inflexibility, ratchet effects, and the inflationary impact of import price changes" (1977), or Blinder's "The supply-shock explanation of the great stagflation revisited" (2013), which belongs to the oil cluster but shares references with the other clusters, are such bridges. In a co-citation network, they are works used by several literatures.
- *Non-hub kinless* (R4): ties spread almost evenly over all clusters, which can mark an outlier as well as a bridge. In the example, the only one, Nitzan's "Regimes of differential accumulation: mergers, stagflation and the logic of globalization" (2001), shares references with five clusters in almost equal parts (17 to 20% of its tie weight each) and is below the average of its own cluster for within-module degree: it is tied to everybody a little, and to nobody much.
- *Peripheral* (R1, R2): all or most ties stay within the cluster. R1 (participation below 0.05) are documents whose ties all stay in their own cluster, such as Phelps's "Commodity-supply shock and full-employment monetary policy" (1978), in the cluster on imports.

The same classification means other things for other nodes. For authors coupled through their references (see `build_dynamic_networks()`), a connector is an author who cites several literatures, a hub an author who cites like most of the group. For networks that are not bibliographic, the roles describe how much each node holds its community together (hubs) and how much it links communities (connectors).

Cautions. (1) Roles depend on the clusters, hence on the algorithm and on the resolution: on the stagflation data, the mean participation coefficient is 0.31, 0.39 and 0.52 for Leiden resolutions of 0.75, 1 and 1.5, because smaller clusters leave more ties between clusters. Check that the nodes

you interpret keep their role when the resolution changes a little. (2) The thresholds of Guimerà and Nunes Amaral (2005) come from metabolic networks. Projected networks are dense (here, a quarter of the pairs of documents are tied) and most nodes have a participation above 0.05: R2 is the usual role, and hubs are rare. (3) With the weights of similarity measures, long bibliographies and well-cited references influence the weights (see `build_dynamic_networks()`). (4) A role is a position in a network, not a judgement on the importance or the quality of a document; reading the document is necessary.

The original measures count edges; here they use the edge weights (strengths), which is the natural generalisation for weighted projected networks. The graph is treated as undirected. The thresholds were calibrated on biological networks: in small bibliometric networks, few nodes pass $z \geq 2.5$, and the measures are best read as continuous indicators (who are the central members of a literature, which articles bridge two literatures) rather than as categories.

Value

The same object as `graphs` with four new node columns: `within_module_degree` (κ_i), `within_module_z` (z_i, 0 when all members of the cluster have the same κ), `participation_coeff` (P_i, 0 for isolated nodes) and `roles` (the role).

References

Guimerà R, Nunes Amaral LA (2005). "Functional Cartography of Complex Metabolic Networks." *Nature*, **433**(7028), 895–900. doi:10.1038/nature03288.

Examples

```
network <- build_network(
  nodes = subset(Nodes_stagflation, source_type == "Stagflation"),
  directed_edges = Ref_stagflation,
  source_id = "source_id",
  target_id = "target_id",
  method = "similarity",
  edges_threshold = 1
)
network <- add_clusters(
  network,
  clustering_method = "leiden",
  objective_function = "modularity",
  seed = 123,
  verbose = FALSE
)

network <- add_node_roles(network, cluster_id = "cluster_leiden")
table(tidygraph::as_tibble(network, active = "nodes")$roles)
```

Authors_stagflation *Authors of the documents explaining the 1970s US stagflation*

Description

All the authors of the documents of Nodes_stagflation that explain the stagflation (Nodes_stagflation only gives the first author), from the corpus of Goutsmedt (2021). It can be used to build networks of authors.

Usage

```
Authors_stagflation
```

Format

A data frame with 231 rows and 3 variables:

source_id Identifier of the document.

author_name Name of the author.

author_order Rank of the author in the list of authors of the document.

Source

Goutsmedt (2021).

References

Goutsmedt A (2021). "From the Stagflation to the Great Inflation: Explaining the US Economy of the 1970s." *Revue d'Économie Politique*, **131**(3), 557–582. https://www.cairn.info/revue-d-economie-politique-2021-3-page-557.htm.

build_dynamic_networks *Build one network, or one network per time window, from bipartite links*

Description

[Experimental]

The first step of the `networkflow` pipeline. Research data rarely come as a network: they come as a table of links between two kinds of entities, such as articles and the references they cite, authors and the papers they sign, or accounts and the hashtags they use. These functions *project* this two-mode (bipartite) relation onto the first kind of entity: two `source_id` entities (e.g. two articles) are linked when they share `target_id` entities (e.g. references).

- `build_network()` builds **one network** from the whole table.
- `build_dynamic_networks()` builds **one network per time window** (`time_variable` and `time_window` are required), which is the starting point of a dynamic analysis. It is `build_network()` repeated on slices of the corpus, with a few more arguments for time.

Write $a_{ij} = 1$ when source i is linked to target j. The raw strength of the tie between sources i and k is the number of targets they share,

$$c_{ik} = \sum_j a_{ij} a_{kj} = |R_i \cap R_k|,$$

with R_i the set of targets of i. When sources are citing documents and targets their references, this is bibliographic coupling (Kessler 1963). Swapping the roles (cited documents as `source_id`, citing documents as `target_id`) gives a co-citation network (Small 1973).

Raw counts favour sources with many targets (articles with long bibliographies), and most pairs share a target or two by chance. The single `method` argument chooses how this problem is answered, among two families:

- a **similarity measure** (`"angle"`, `"strength"`, `"similarity"`, `"jaccard"`, `"vladutz"`) normalises the counts, as implemented in biblionetwork, and the pairs sharing at least `edges_threshold` targets are kept, with their weight;
- a **statistical backbone** (`"sdsm"`, `"fdsm"`, `"fixedfill"`, `"fixedrow"`, `"fixedcol"`) keeps only the ties that are stronger than expected under a null model of random co-occurrence, as implemented in the backbone package (Neal 2014, 2022).

There is no default `method`: the choice changes the network, so it must be explicit. With `verbose = TRUE` the function says which method it uses and where it comes from. (The arguments `projection_method`, `cooccurrence_method` and `model` of earlier versions are replaced by `method` and give an error that explains the translation.)

Usage

```
build_dynamic_networks(
  nodes,
  directed_edges,
  source_id,
  target_id,
  document_id = NULL,
  projection_on = c("source", "target"),
  method,
  edges_threshold = 1,
  alpha = NULL,
  backbone_args = list(),
  time_variable = NULL,
  time_in = c("nodes", "edges"),
  time_unit = NULL,
  time_window = NULL,
  window_step = NULL,
  compute_size = FALSE,
```

```
  keep_singleton = FALSE,
  filter_components = FALSE,
  ...,
  verbose = TRUE
)
```

Arguments

nodes	A data frame with one row per source entity (e.g. one row per article) and its metadata (authors, year, title...). Not modified.
directed_edges	A data frame of bipartite links, one row per source_id -> target_id link (e.g. article -> cited reference). Duplicated links are counted once. Not modified.
source_id	Character scalar, the column identifying the entities that become the nodes of the network. It must exist in both nodes and directed_edges, with one row per value in nodes.
target_id	Character scalar, the column of directed_edges with the entities that create the links (e.g. references).
document_id	Optional column of directed_edges identifying the document through which each link exists. With it, source_id entities are not documents but *entities that publish documents* (authors, journals, institutions) and a reference cited by several documents of an entity counts several times, following biblionetwork::entity_network(). Only for the similarity measures.
projection_on	"source" (default) links the source_id entities that share target_id entities (bibliographic coupling: articles sharing references). "target" links the target_id entities that share source_id entities (co-citation: cited documents linked when they are cited by the same documents). With "target", nodes is keyed by target_id.
method	Required, no default. How the ties between sources are built; the origin of each method is given in the details. One of • the similarity measures of biblionetwork (biblionetwork::biblio_coupling(method =), or biblionetwork::entity_network() with document_id): "angle", "strength", "similarity", "jaccard", "vladutz"; • the null models of the statistical backbones of backbone::backbone_from_projection(): "sdsm", "fdsm", "fixedfill", "fixedrow", "fixedcol". They need alpha.
edges_threshold	For the similarity measures, the minimum number of shared targets for two sources to be linked (1 by default). Ignored by the backbone methods.
alpha	For the backbone methods, the significance level of the test of each tie (e.g. 0.05). Lower values keep fewer edges. Required for them, ignored by the similarity measures.
backbone_args	Optional list of further arguments for backbone::backbone_from_projection() (e.g. mtc = "holm" to correct for multiple tests). alpha given here overrides the argument of the function.

time_variable	Name of a column with the time of each node (or of each link, with time_in = "edges"): a number (e.g. a year), a Date, a POSIXct, or an ISO date written as text ("2001-03", "2001-03-15"). Years written as text ("2001") are numbers. Required by build_dynamic_networks() (without time, use build_network()).
time_in	"nodes" (default): time_variable is a column of nodes and a window selects the nodes of the period. "edges": time_variable is a column of directed_edges (e.g. the year of the citing document) and a window selects the links of the period, the nodes being those with a link in the window. Use "edges" for co-citation, whose period is the one of the citing documents, and for entities without a date of their own (authors).
time_unit	For dates only: the unit in which windows are counted, "year" (default), "quarter", "month", "week" (starting on Mondays) or "day". For numbers, the unit is the one of the variable and time_unit must stay NULL.
time_window	Whole number, the width of each window in time_units (or in units of the variable): 10 for ten years, 12 with time_unit = "month" for a year. Required by build_dynamic_networks().
window_step	Optional whole number, the number of units between the starts of two windows. NULL (default) gives successive windows (window_step = time_window); 1 slides the window one unit at a time (1975-1984, 1976-1985...); 3 with time_unit = "month" and a window of 12 gives a yearly window every quarter.
compute_size	Logical. If TRUE, adds a node_size column: the number of times each node is a target of the sources of the same window (for a coupling network, the number of citations received within the corpus). It requires nodes to contain the target_id column.
keep_singleton	Logical. If FALSE (default), nodes without any edge are removed.
filter_components	Logical. If TRUE, keeps only the main component(s) with filter_components().
...	Arguments passed to filter_components() (e.g. nb_components).
verbose	Logical. If TRUE (default), reports the steps.

Details

The arguments to choose, case by case. projection_on says which side of the links becomes the nodes, and method says how the ties between them are built:

You want	Function and main argument
A network of the whole corpus, ties weighted by a similarity measure (bibliographic coupling)	build_network(method = "
The same, with only the ties unlikely under a null model	build_network(method = "
One such network per period	build_dynamic_networks(
Cited works linked when cited together (co-citation)	projection_on = "target"
Authors, journals or institutions linked through the references of their documents	document_id = the column
Any other two-mode table (accounts and hashtags, members and boards)	the same arguments: nothing

The two families of method**.** They do not use the same arguments:

	Similarity measures	Sta
`method`	`"angle"`, `"strength"`, `"similarity"`, `"jaccard"`, `"vladutz"`	`"s`
Idea	normalise the number of shared targets, then keep the pairs sharing at least `edges_threshold` targets	tes
Edges	weighted (`weight`)	un
Arguments used	`edges_threshold`, `document_id`	`al`
Ignored	`alpha`, `backbone_args`	`ed`
Cost	fast, sparse matrices: seconds for tens of thousands of documents	a t
Computed by	biblionetwork	the

Where the methods come from. The names are those of the packages that compute them, so that the documentation of each method can be found there:

`method`	What it is	Origin
`"angle"`	cosine of the lists of targets	(Sen and Gan 1983)
`"jaccard"`	shared targets over the union of the lists	(Sen and Gan 1983)
`"strength"`	shared targets weighted by their rarity	(Shen et al. 2019)
`"similarity"`	idf-weighted cosine	(Shen et al. 2019)
`"vladutz"`	Vladutz-Cook formula	(Vladutz and Cook
`"sdsm"`, `"fdsm"`, `"fixedfill"`, `"fixedrow"`, `"fixedcol"`	null models of random bipartite networks	(Neal 2014, 2022; D

Similarity measures. With $L_i = |R_i|$ the number of targets of source i, N the number of sources and f_j the number of sources linked to target j, the five measures computed by `biblionetwork` (through `biblionetwork::biblio_coupling()`, or `biblionetwork::entity_network()` with `document_id`) are:

- `"angle"`, the cosine of Sen and Gan (1983):

$$w_{ik} = \frac{|R_i \cap R_k|}{\sqrt{L_i L_k}};$$

- `"jaccard"`, the relative coupling strength of Sen and Gan (1983), a Jaccard index: the shared targets over the *union* of the two lists,

$$w_{ik} = \frac{|R_i \cap R_k|}{|R_i \cup R_k|}.$$

  Like the angle it only counts, but it is never above it and penalises more the pairs whose lists have very different lengths;

- `"strength"` (Shen et al. 2019), where a shared target counts less when it is linked to many sources, in the spirit of an inverse document frequency:

$$w_{ik} = \frac{\sum_{j \in R_i \cap R_k} \log(N/f_j)}{L_i L_k};$$

- `"similarity"`, the idf-weighted cosine of Shen et al. (2019): each source is described by a vector with one component per target, equal to $\sqrt{\log(N/f_j)}$, and the weight is the cosine of the vectors of two sources, with $S_i = \sum_{j \in R_i} \log(N/f_j)$ the weighted size of the bibliography:

$$w_{ik} = \frac{\sum_{j \in R_i \cap R_k} \log(N/f_j)}{\sqrt{S_i S_k}}.$$

Like the coupling angle, it lies between 0 and 1; like the coupling strength, it discounts the targets linked to many sources;

- `"vladutz"`, the formula of Vladutz and Cook (1984) reported by Shen et al. (2019), the ancestor of the coupling strength, with a much steeper discount ($1/f_j$ instead of $\log(N/f_j)$), so that it is almost driven by the rarest targets:

$$w_{ik} = \frac{4}{L_i + L_k} \sum_{j \in R_i \cap R_k} \frac{1}{f_j}.$$

The measures fall into two families: those that only count the shared targets (angle, Jaccard) and those that give more weight to rare targets (strength, similarity, Vladutz-Cook). The angle and the Jaccard index give very similar rankings of the pairs in practice (Pearson correlation of 0.956 in the corpus of Shen et al. (2019)), whereas the Vladutz-Cook formula departs markedly from the others there (correlations of about 0.6 to 0.7). It is provided mainly for comparison and replication. See the biblionetwork documentation of each measure (`coupling_angle()`, `coupling_jaccard()`, `coupling_strength()`, `coupling_similarity()`, `coupling_vladutz()`) for the intuition behind them. (Versions 0.1.1 and earlier of `biblionetwork` computed the similarity wrongly: `biblionetwork` 0.2.0 or later is required.)

Targets linked to a single source cannot create a tie and are dropped before the computation. The threshold applies to the raw count $|R_i \cap R_k|$, not to the weight: `edges_threshold = 2` keeps the pairs of articles that share at least two references, a common choice to remove ties that one shared classic is enough to create. The measures differ in how they treat very cited targets and long target lists; in practice, clusters are often robust to the choice, but it is worth checking on one's own data.

Statistical backbones. Instead of fixing a threshold, the backbone approach asks, for each pair, whether the observed number of shared targets is larger than in random bipartite networks that preserve some properties of the observed one: the degrees of sources and targets on average (`"sdsm"`, the stochastic degree sequence model) or exactly (`"fdsm"`), or only the total number of links (`"fixedfill"`) (Neal 2014; Domagalski et al. 2021). Edges are kept when the test is significant at level `alpha`; the resulting network is unweighted. This is slower, but each edge can be read as a tie unlikely to be due to chance.

Time windows. Time is counted in whole units: the values of a numeric variable, or years, quarters, months, weeks or days for dates. A window starting at unit s and w units wide (`time_window`) gathers the nodes (or links) with $s \leq t < s + w$, and the network is computed as if the corpus were limited to them. A new window starts every `window_step` units: successive windows (the default) give independent snapshots, a step of 1 slides the window and smooths the evolution of the networks, at the price of many networks sharing most of their nodes. The last window may be shorter than the others when the period is not a multiple of the step (a warning is given); windows are labelled by their first and last unit (`"1975-1984"`, `"2001-01 to 2001-12"`). The choice of the window is substantive: long enough for each network to have enough nodes (a few dozen at least) to show a structure, and short enough for changes to be visible. For documents that follow each other quickly (central bank statements, news, posts), months or weeks make sense, with a window wide enough to contain enough documents.

Beyond bibliographic coupling. Nothing in the function is specific to citations. The two sides of the links can be any two kinds of entities:

- *Co-citation* (Small 1973): with `projection_on = "target"`, cited documents are linked when the same documents cite them. Windows should follow the citing documents: put their year in `directed_edges` and use `time_in = "edges"`.

- *Author (or journal, institution) coupling* (Zhao and Strotmann 2008): with `document_id`, authors are linked when the documents they sign cite the same references.
- *Co-authorship and other co-occurrences*: authors and the documents they sign, accounts and the hashtags they use, members and the boards they sit on: any two-mode table.

Identifiers of references. Every measure assumes that one work has one identifier. With raw bibliographic data, the same reference is often written in several ways (spelling, abbreviations, missing years), each spelling becomes a different `target_id`, and two documents citing the same work in different ways no longer share it. This matters most for co-citation, where cited works are the nodes: a work split into several spellings is split into several nodes, each with a fraction of its citations. It matters less when the database gives an identifier to each cited reference (e.g. the identifiers of Scopus, OpenAlex or Web of Science), but even then references that the database could not match are missing or duplicated. Clean and match references before building networks, and check the most cited nodes for duplicates.

Value

A `tbl_graph` (undirected) if `time_variable` is `NULL`; otherwise a list of `tbl_graphs` named after their window (e.g. `"1975-1984"`), in chronological order, whose nodes have a `time_window` column. Edges have a `weight` column for the structured method.

References

Domagalski R, Neal ZP, Sagan B (2021). "Backbone: An R Package for Extracting the Backbone of Bipartite Projections." *PLOS ONE*, **16**(1), e0244363. doi:10.1371/journal.pone.0244363.

Kessler MM (1963). "Bibliographic Coupling between Scientific Papers." *American Documentation*, **14**(1), 10–25. doi:10.1002/asi.5090140103.

Neal Z (2014). "The Backbone of Bipartite Projections: Inferring Relationships from Co-Authorship, Co-Sponsorship, Co-Attendance and Other Co-Behaviors." *Social Networks*, **39**, 84–97. doi:10.1016/j.socnet.2014.06.001.

Neal ZP (2022). "backbone: An R Package to Extract Network Backbones." *PLOS ONE*, **17**(5), e0269137. doi:10.1371/journal.pone.0269137.

Sen SK, Gan SK (1983). "A Mathematical Extension of the Idea of Bibliographic Coupling and Its Applications." *Annals of Library Science and Documentation*, **30**(2), 78–82.

Shen S, Zhu D, Rousseau R, Su X, Wang D (2019). "A Refined Method for Computing Bibliographic Coupling Strengths." *Journal of Informetrics*, **13**(2), 605–615. doi:10.1016/j.joi.2019.01.012.

Small H (1973). "Co-Citation in the Scientific Literature: A New Measure of the Relationship between Two Documents." *Journal of the American Society for Information Science*, **24**(4), 265–269. doi:10.1002/asi.4630240406.

Vladutz G, Cook J (1984). "Bibliographic Coupling and Subject Relatedness." *Proceedings of the American Society for Information Science*, **21**, 204–207.

Zhao D, Strotmann A (2008). "Author Bibliographic Coupling: Another Approach to Citation-Based Author Knowledge Network Analysis." In *Proceedings of the American Society for Information Science and Technology*, volume 45(1), 1–10. https://asistdl.onlinelibrary.wiley.com/doi/full/10.1002/meet.2008.1450450292.

See Also

build_network() for a single network; biblionetwork::biblio_coupling() and the measures it offers (biblionetwork::coupling_angle(), biblionetwork::coupling_jaccard(), biblionetwork::coupling_st biblionetwork::coupling_similarity(), biblionetwork::coupling_vladutz()), biblionetwork::entity_networ backbone::backbone_from_projection(). The next steps of the pipeline are add_clusters() and merge_dynamic_clusters().

Examples

```
nodes <- subset(Nodes_stagflation, source_type == "Stagflation")

# Bibliographic coupling networks of the articles explaining the 1970s
# stagflation, one per decade
networks <- build_dynamic_networks(
  nodes = nodes,
  directed_edges = Ref_stagflation,
  source_id = "source_id",
  target_id = "target_id",
  time_variable = "source_year",
  time_window = 13,
  method = "similarity",
  edges_threshold = 1,
  verbose = FALSE
)
names(networks)
networks[[1]]

# Co-citation: cited documents linked when the same documents cite them,
# with windows on the year of the citing documents
citations <- merge(
  transform(Ref_stagflation[, c("source_id", "target_id")],
            source_id = as.character(source_id), target_id = as.character(target_id)),
  nodes[, c("source_id", "source_year")],
  by = "source_id"
)
cited <- Ref_stagflation[!duplicated(Ref_stagflation$target_id),
                         c("target_id", "target_author", "target_year")]
cocitation <- build_dynamic_networks(
  nodes = transform(cited, target_id = as.character(target_id)),
  directed_edges = citations,
  source_id = "source_id",
  target_id = "target_id",
  projection_on = "target",
  time_in = "edges",
  time_variable = "source_year",
  time_window = 13,
```

```
  method = "angle",
  edges_threshold = 2,
  compute_size = TRUE,
  verbose = FALSE
)
cocitation[[1]]

# Authors linked by the references of the documents they sign
by_author <- merge(
  transform(Authors_stagflation, source_id = as.character(source_id)),
  citations, by = "source_id"
)
author_coupling <- build_network(
  nodes = data.frame(author_name = unique(by_author$author_name)),
  directed_edges = by_author,
  source_id = "author_name",
  target_id = "target_id",
  document_id = "source_id",
  method = "strength",
  edges_threshold = 2
)
author_coupling


# Statistical backbone: keep the ties that random networks with the same
# degrees would rarely produce
if (requireNamespace("backbone", quietly = TRUE)) {
backbones <- build_dynamic_networks(
  nodes = nodes,
  directed_edges = Ref_stagflation,
  source_id = "source_id",
  target_id = "target_id",
  time_variable = "source_year",
  time_window = 20,
  method = "sdsm",
  alpha = 0.05,
  backbone_args = list(mtc = "holm"),
  verbose = FALSE
)
}
```

`build_network` *Build a single network from bipartite links*

Description

[Experimental]

build_network() projects a two-mode relation (e.g. articles and the references they cite) onto a one-mode network of the source entities, for the whole corpus at once. It is build_dynamic_networks() without time windows: see that function for the measures, the thresholds and the literature behind them.

Usage

```
build_network(
  nodes,
  directed_edges,
  source_id,
  target_id,
  document_id = NULL,
  projection_on = c("source", "target"),
  method,
  edges_threshold = 1,
  alpha = NULL,
  backbone_args = list(),
  compute_size = FALSE,
  keep_singleton = FALSE,
  filter_components = FALSE,
  ...,
  verbose = TRUE
)
```

Arguments

nodes	A data frame with one row per source entity (e.g. one row per article) and its metadata (authors, year, title...). Not modified.
directed_edges	A data frame of bipartite links, one row per source_id -> target_id link (e.g. article -> cited reference). Duplicated links are counted once. Not modified.
source_id	Character scalar, the column identifying the entities that become the nodes of the network. It must exist in both nodes and directed_edges, with one row per value in nodes.
target_id	Character scalar, the column of directed_edges with the entities that create the links (e.g. references).
document_id	Optional column of directed_edges identifying the document through which each link exists. With it, source_id entities are not documents but *entities that publish documents* (authors, journals, institutions) and a reference cited by several documents of an entity counts several times, following biblionetwork::entity_network(). Only for the similarity measures.
projection_on	"source" (default) links the source_id entities that share target_id entities (bibliographic coupling: articles sharing references). "target" links the target_id entities that share source_id entities (co-citation: cited documents linked when they are cited by the same documents). With "target", nodes is keyed by target_id.
method	Required, no default. How the ties between sources are built; the origin of each method is given in the details. One of

- the similarity measures of biblionetwork (`biblionetwork::biblio_coupling(method = )`, or `biblionetwork::entity_network()` with `document_id`): `"angle"`, `"strength"`, `"similarity"`, `"jaccard"`, `"vladutz"`;
- the null models of the statistical backbones of `backbone::backbone_from_projection()`: `"sdsm"`, `"fdsm"`, `"fixedfill"`, `"fixedrow"`, `"fixedcol"`. They need `alpha`.

`edges_threshold`
: For the similarity measures, the minimum number of shared targets for two sources to be linked (1 by default). Ignored by the backbone methods.

`alpha`
: For the backbone methods, the significance level of the test of each tie (e.g. 0.05). Lower values keep fewer edges. Required for them, ignored by the similarity measures.

`backbone_args`
: Optional list of further arguments for `backbone::backbone_from_projection()` (e.g. `mtc = "holm"` to correct for multiple tests). `alpha` given here overrides the argument of the function.

`compute_size`
: Logical. If `TRUE`, adds a `node_size` column: the number of times each node is a target of the sources of the same window (for a coupling network, the number of citations received within the corpus). It requires `nodes` to contain the `target_id` column.

`keep_singleton`
: Logical. If `FALSE` (default), nodes without any edge are removed.

`filter_components`
: Logical. If `TRUE`, keeps only the main component(s) with `filter_components()`.

`...`
: With `filter_components = TRUE`, arguments of `filter_components()` (e.g. `nb_components`).

`verbose`
: Logical. If `TRUE` (default), reports the steps.

Details

The cases (which arguments for which kind of network), the methods and where they come from are described in the Details of `build_dynamic_networks()`. With `verbose = TRUE` (default), the function says which method it applies.

Value

A `tbl_graph`, undirected, with the columns of `nodes` as node attributes and, for the structured method, a `weight` edge column.

See Also

`build_dynamic_networks()`; `filter_components()`, `add_clusters()` for the next steps.

Examples

```
nodes <- subset(Nodes_stagflation, source_type == "Stagflation")

network <- build_network(
  nodes = nodes,
```

```
  directed_edges = Ref_stagflation,
  source_id = "source_id",
  target_id = "target_id",
  method = "similarity",
  edges_threshold = 2
)
network
```

`color_networks` *Colour the nodes and edges of networks, or the clusters of an alluvial*

Description

[Experimental]

`color_networks()` gives a colour to each category of a node column (typically a cluster column) and stores it in a `color` column of the nodes. Each edge gets a colour mixing the colours of its two ends, so that edges within a cluster take the colour of the cluster and edges between clusters a blend of both.

`color_alluvial()` does the same for the table of an alluvial plot created with `networks_to_alluv()`.

Usage

```
color_networks(
  graphs,
  column_to_color,
  color = NULL,
  unique_color_across_list = FALSE,
  verbose = TRUE
)

color_alluvial(alluv_dt, column_to_color, color = NULL, verbose = TRUE)
```

Arguments

`graphs`	A `tbl_graph` or a list of `tbl_graph`s.
`column_to_color`	Character scalar, the column with the categories to colour, e.g. `cluster_leiden` (`add_clusters()`) or `dynamic_cluster_leiden` (`merge_dynamic_clusters()`).
`color`	The colours: `NULL` (default palettes, see the details), a character vector of colours, given to the categories by decreasing number of nodes, or a data frame whose first column holds the categories and the second column, named `color`, their colours.
`unique_color_across_list`	Logical, for a list of graphs. `FALSE` (default) gives a category the same colour in every graph: use it for dynamic clusters. `TRUE` treats the same value in two graphs as two categories (cluster "01" of two windows gets two colours): use it for clusters detected separately in each window.

verbose	Logical. If TRUE (default), reports when colours are recycled or truncated.
alluv_dt	A data frame created with networks_to_alluv().

Details

Giving the same colour to a cluster in every figure (network plots, alluvial, tables) is what lets a reader follow it. Pass the same data frame of colours to color_networks() and color_alluvial() for this.

Without color, the function uses the ggplot2 palette (without black) for up to 7 categories, completed by the Okabe-Ito palette (without black and grey) for up to 14; beyond 14 categories, the colours are recycled and several clusters share a colour. For many clusters, give a palette explicitly (e.g. from the scico or paletteer packages), or colour only the largest clusters and give the others a neutral grey.

A vector with fewer colours than categories is recycled, a longer one is truncated. A data frame of colours that misses some categories leaves their colour missing.

Value

color_networks(): the same object as graphs with a color column for nodes and edges. color_alluvial(): alluv_dt with a color column.

Examples

```
network <- build_network(
  nodes = subset(Nodes_stagflation, source_type == "Stagflation"),
  directed_edges = Ref_stagflation,
  source_id = "source_id",
  target_id = "target_id",
  method = "similarity",
  edges_threshold = 1
)
network <- add_clusters(
  network,
  clustering_method = "leiden",
  objective_function = "modularity",
  seed = 123,
  verbose = FALSE
)

network <- color_networks(network, column_to_color = "cluster_leiden")
unique(tidygraph::as_tibble(network, active = "nodes")[c("cluster_leiden", "color")])
```

Edges_coupling *Edges of a ready-made bibliographic coupling network on the stagflation*

Description

The bibliographic coupling edges between the documents of Nodes_coupling, computed with `biblionetwork::coupling_a "source_id", "target_id")` (coupling angle, see `build_dynamic_networks()`).

Usage

```
Edges_coupling
```

Format

A data frame with 2,593 rows and 5 variables:

from Identifier of the first document, as a character string.

to Identifier of the second document, as a character string.

weight Coupling angle between the two documents (see `biblionetwork::coupling_angle()`).

Source Copy of `from`.

Target Copy of `to`.

Source

Built from Ref_stagflation by `inst/data-raw/creating_network_data.R`.

---

extract_tfidf *Find the terms that distinguish groups of nodes (tf-idf)*

Description

[Experimental]

Clusters are defined by the structure of the network, not by the content of the nodes. To interpret them, it helps to know which words are specific to each cluster. `extract_tfidf()` extracts the words and n-grams of one or several text columns (e.g. titles, abstracts) and returns, for each group (e.g. each cluster), the terms with the highest tf-idf (Spärck Jones 1972; Salton and Buckley 1988):

$$\mathrm{tf}_{td} = \frac{n_{td}}{\sum_u n_{ud}}, \qquad \mathrm{idf}_t = \log \frac{D}{\mathrm{df}_t}, \qquad \text{tf-idf}_{td} = \mathrm{tf}_{td} \times \mathrm{idf}_t,$$

with n_{td} the count of term t in group d, D the number of groups and df_t the number of groups containing t. A term present in every group has a tf-idf of 0.

Usage

```
extract_tfidf(
  data,
  text_columns,
  grouping_columns,
  grouping_across_list = FALSE,
  n_gram = 2L,
  stopwords_type = "smart",
  stopwords_vector = NULL,
  clean_word_method = c("lemmatize", "stemming", "none"),
  ngrams_filter = 5L,
  nb_terms = 5L
)
```

Arguments

`data`	A `tbl_graph`, a list of `tbl_graph`s (their node tables are stacked) or a data frame.
`text_columns`	Character vector, the column(s) with text. Several columns are pasted together, separated by ". ".
`grouping_columns`	Character vector, the column(s) defining the groups, the "documents" of the tf-idf (e.g. `"cluster_leiden"`).
`grouping_across_list`	Logical. For a list of graphs whose group identifiers repeat across graphs (cluster "01" exists in every window), `TRUE` adds the name of the graph to the groups, so that each cluster of each window is a separate document. Not needed for identifiers that are already unique across the list, such as dynamic clusters.
`n_gram`	Integer, the maximum length of n-grams (2 by default: words and pairs of words).
`stopwords_type`	Character scalar, the stop word list of `tidytext::get_stopwords()` (`"smart"` by default).
`stopwords_vector`	Optional character vector of stop words, used instead of `stopwords_type`.
`clean_word_method`	`"lemmatize"`, `"stemming"` (with the `textstem` package) or `"none"` (chosen when no method is given).
`ngrams_filter`	Integer, the minimum number of occurrences of a term in the whole corpus (5 by default).
`nb_terms`	Integer, the number of terms returned per group (5 by default).

Details

The texts are lower-cased and split into n-grams of 1 to `n_gram` words by `tidytext::unnest_tokens()` (Silge and Robinson 2017). Each word is lemmatised or stemmed if requested; n-grams containing a stop word or a number are dropped, as are terms rarer than `ngrams_filter` in the corpus and

terms appearing only once in a group, which would otherwise reach the top of small groups by accident.

The choice of the groups decides what "specific" means: computed on the clusters of one network, tf-idf finds what distinguishes a cluster from the other clusters of the same period; computed on the stacked windows with `grouping_across_list = TRUE`, it also distinguishes a cluster from its own past and future.

Lemmatising and stemming are optional and need the `textstem` package. For larger corpora or other weighting schemes, the text-mining functions of the mintR package (https://codeberg.org/agoutsmedt/mintR) can be used on the node table.

Value

A `data.table` with one row per group and term, for the `nb_terms` best terms of each group: the grouping columns, `term`, `n` (count of the term in the group), `tf`, `idf` and `tf_idf`.

References

Salton G, Buckley C (1988). "Term-Weighting Approaches in Automatic Text Retrieval." *Information Processing & Management*, **24**(5), 513–523.

Silge J, Robinson D (2017). *Text Mining with R: A Tidy Approach*. O'Reilly.

Spärck Jones K (1972). "A Statistical Interpretation of Term Specificity and Its Application in Retrieval." *Journal of Documentation*, **28**(1), 11–21.

Examples

```
network <- build_network(
  nodes = subset(Nodes_stagflation, source_type == "Stagflation"),
  directed_edges = Ref_stagflation,
  source_id = "source_id",
  target_id = "target_id",
  method = "similarity",
  edges_threshold = 1
)
network <- add_clusters(
  network,
  clustering_method = "leiden",
  objective_function = "modularity",
  seed = 123,
  verbose = FALSE
)

extract_tfidf(
  network,
  text_columns = "source_title",
  grouping_columns = "cluster_leiden",
  n_gram = 2,
  clean_word_method = "none",
  nb_terms = 3
```

```
)
```

`filter_components` *Keep the main component(s) of networks*

Description

[Experimental]

A projected network is often made of one large connected component and of small islands: a few articles that share references only with each other, or isolated pairs. Community detection and layouts are hard to read on such fragments, and many studies keep only the main component. `filter_components()` numbers the (weakly) connected components by decreasing size, keeps the `nb_components` largest, and warns when the first component removed is not negligible.

Usage

```
filter_components(
  graphs,
  nb_components = 1,
  threshold_alert = 0.05,
  keep_component_columns = FALSE
)
```

Arguments

`graphs`	A `tbl_graph` or a list of `tbl_graphs` (e.g. the output of `build_dynamic_networks()`).
`nb_components`	Integer, the number of components to keep, starting from the largest (1 by default).
`threshold_alert`	Number between 0 and 1. If the largest removed component gathers more than this share of the nodes (0.05 by default), a warning is given: removing it would change the object of study.
`keep_component_columns`	Logical. If `TRUE`, keeps the columns `components_att` (component rank, 1 being the largest) and `size_components` (share of the nodes in the component).

Details

Dropping small components is a choice, not a technicality: the removed documents may be a marginal literature, or a literature that cites sources absent from the data. Look at what is removed before discarding it, especially when the warning is triggered. For a list of graphs, the filter is applied to each graph separately, so a node may be kept in one window and removed in another.

Value

The same object as `graphs` (a `tbl_graph` or a list of `tbl_graphs`) restricted to the nodes of the kept components.

See Also

build_network() and build_dynamic_networks(), which can call this function with filter_components = TRUE.

Examples

```
network <- build_network(
  nodes = subset(Nodes_stagflation, source_type == "Stagflation"),
  directed_edges = Ref_stagflation,
  source_id = "source_id",
  target_id = "target_id",
  method = "similarity",
  edges_threshold = 2
)
main_component <- filter_components(network, nb_components = 1)
c(before = igraph::vcount(network), after = igraph::vcount(main_component))
```

---

layout_networks *Compute the coordinates of nodes*

Description

[Experimental]

A network has no natural position on a page. Force-directed layouts place nodes so that tied nodes are close and untied nodes are pushed apart, so that clusters appear as dense regions: Kamada-Kawai ("kk") draws graph distances as geometric distances (Kamada and Kawai 1989), Fruchterman-Reingold ("fr") balances attraction along edges and repulsion between all nodes (Fruchterman and Reingold 1991). layout_networks() computes such coordinates with ggraph::create_layout(), which gives access to the layouts of igraph and graphlayouts, and stores them in the node table. Two layouts that are not in ggraph are added: "forceatlas2" (Jacomy et al. 2014), the force-directed layout of Gephi, and "umap" (McInnes et al. 2018), which embeds the distances between nodes with the UMAP algorithm.

For a list of networks, the coordinates of a node may jump from one window to the next, which makes the plots hard to compare. With compute_dynamic_coordinates = TRUE, the coordinates of window $t-1$ are used as the starting positions of the layout of window t, so that the nodes present in both windows tend to stay in the same region.

Usage

```
layout_networks(
  graphs,
  node_id,
  layout,
  compute_dynamic_coordinates = FALSE,
  save_coordinates = FALSE,
  ...
)
```

Arguments

graphs	A tbl_graph or a list of tbl_graphs.
node_id	Character scalar, the node column with a unique identifier, used to join the coordinates.
layout	Character scalar, a layout accepted by ggraph::create_layout() (e.g. "kk", "fr", "stress"), or "forceatlas2" or "umap" (see the details).
compute_dynamic_coordinates	Logical. For a list of graphs, if TRUE, each layout starts from the coordinates of the previous window (see the details).
save_coordinates	Logical. If TRUE, the coordinates are also stored in x_{layout} and y_{layout} (e.g. x_kk, y_kk), so that they are not lost when another layout is computed.
...	Other arguments of the layout (see ggraph::create_layout() and the igraph or graphlayouts function behind it; the arguments of "forceatlas2" and "umap" are given in the details).

Details

Force-directed layouts are stochastic and sensitive to the starting positions: set a seed with set.seed() before calling the function to reproduce a figure. Distances in such plots are only roughly interpretable: two clusters drawn side by side are more tied than two clusters at opposite ends, but distances should not be measured.

Weights. igraph uses the weight edge column automatically, but not always in the same way: Fruchterman-Reingold ("fr") reads weights as attraction (heavier edges pull nodes closer), while Kamada-Kawai ("kk") reads them as distances, so that similarity weights push the most similar nodes apart and weakly tied nodes to the margins. With similarity weights, prefer "fr", or pass weights = NA to ignore them.

ForceAtlas2 (layout = "forceatlas2"). A force-directed layout of the same family as Fruchterman-Reingold, designed for the exploration of networks (Jacomy et al. 2014). Edges attract their ends in proportion to their weight and to the distance; all nodes repel each other in proportion to the product of their degrees plus one, so that hubs push the others away and clusters separate; a gravity keeps the components around the centre. The speed adapts to the stability of the forces. This is a compact implementation, not the one of Gephi: the repulsion is computed between all the pairs of nodes (no Barnes-Hut approximation), so that the cost grows with the square of the number of nodes: on a Web of Science sample, about 1 minute for 1,000 nodes and 4 to 5 minutes for 2,000, so that "fr" or "umap" are better choices for larger networks. Arguments: iterations (500), scaling_ratio (2, the strength of the repulsion: smaller values give a more compact layout), gravity (1), strong_gravity (FALSE; TRUE keeps disconnected components close), linlog (FALSE; TRUE uses a logarithmic attraction, which separates the clusters more) and weights (the edge column is used by default, divided by its maximum; NA ignores it). Isolated components drift away from the rest: keep the main component with filter_components() first, or use strong_gravity = TRUE.

UMAP (layout = "umap"). UMAP (McInnes et al. 2018) places points in the plane so as to preserve their neighbourhoods. It needs distances between all the pairs of nodes, which a network does not give directly: the distance between two nodes is the length of the shortest path between them, an edge being as long as the inverse of its weight (a strong tie is a short one), and nodes

of different components are set farther apart than any connected pair. The coordinates come from `uwot::umap()`. Arguments: `n_neighbors` (15, the size of the neighbourhood preserved: larger values give a more global view), `min_dist` (0.1, how tightly nodes are packed), `spread` (1) and `weights` (as above), and any other argument of `uwot::umap()`. The package `uwot` must be installed. UMAP often gives compact, well separated clusters, but, like the force-directed layouts, its distances are not to be measured, and the picture is partly built by the choice of `n_neighbors`.

For dynamic coordinates, nodes absent from the previous window start at (0, 0). The starting positions are passed to the layout as its `coords` argument: this works only with layouts that accept it, such as `"fr"` and `"kk"` of `igraph`, and `"forceatlas2"`; `"umap"` ignores them. With starting positions, `igraph`'s Fruchterman-Reingold no longer uses its fast grid approximation: on a Web of Science window of 9,600 nodes, about 80 seconds instead of 1. As the positions are already close to the result, fewer iterations are often enough (e.g. `niter = 100`, passed to `igraph::layout_with_fr()`).

Value

The same object as `graphs`, with node columns `x` and `y` (any existing `x` and `y` are replaced) and, if `save_coordinates = TRUE`, `x_{layout}` and `y_{layout}`.

References

Fruchterman TMJ, Reingold EM (1991). "Graph Drawing by Force-Directed Placement." *Software: Practice and Experience*, **21**(11), 1129–1164. doi:10.1002/spe.4380211102.

Jacomy M, Venturini T, Heymann S, Bastian M (2014). "ForceAtlas2, a Continuous Graph Layout Algorithm for Handy Network Visualization Designed for the Gephi Software." *PLoS ONE*, **9**(6), e98679. doi:10.1371/journal.pone.0098679.

Kamada T, Kawai S (1989). "An Algorithm for Drawing General Undirected Graphs." *Information Processing Letters*, **31**(1), 7–15. doi:10.1016/00200190(89)901026.

McInnes L, Healy J, Melville J (2018). "UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction." doi:10.48550/arXiv.1802.03426. 1802.03426.

Examples

```
network <- build_network(
  nodes = subset(Nodes_stagflation, source_type == "Stagflation"),
  directed_edges = Ref_stagflation,
  source_id = "source_id",
  target_id = "target_id",
  method = "similarity",
  edges_threshold = 2
)
set.seed(1)
network <- layout_networks(network, node_id = "source_id", layout = "fr")
head(tidygraph::as_tibble(network, active = "nodes")[c("source_id", "x", "y")])
```

`merge_diagnostics` *Diagnose the threshold of the matching of clusters across windows*

Description

[Experimental]

`merge_dynamic_clusters()` links the clusters of successive windows when they share a majority of their nodes (`method = "majority"`) or have a large enough Jaccard index (`method = "jaccard"`), and the result depends on the threshold and on the type of similarity. A low threshold links clusters easily and produces a few long-lived dynamic clusters; a high one breaks continuities and produces many short-lived clusters. There is no right value, but there are *unstable* regions, where a small change of the threshold changes the story. `merge_diagnostics()` runs the matching for a grid of thresholds and similarity types and summarises, for each setting, how the dynamic clusters change: how many there are, how long they live, how many window clusters find a predecessor, and how the nodes are spread between long-lived and short-lived clusters. `plot_merge_diagnostics()` draws the result.

Usage

```
merge_diagnostics(
  graphs,
  cluster_id,
  node_id,
  method = c("majority", "jaccard"),
  thresholds = NULL,
  similarity_types = c("complete", "partial"),
  min_length = 3,
  verbose = TRUE
)
```

Arguments

`graphs`	A list of at least two `tbl_graphs` in chronological order, with the clusters of each window (see `add_clusters()`).
`cluster_id`	Character scalar, the node column with the clusters of each window (e.g. `"cluster_leiden"`).
`node_id`	Character scalar, the node column with an identifier that is stable across windows.
`method`	`"majority"` or `"jaccard"`, the rule of `merge_dynamic_clusters()` to diagnose. Required: run the function once for each rule to compare them.
`thresholds`	Numeric vector of thresholds to try, in $(0.5, 1]$ for `"majority"` and $(0, 1]$ for `"jaccard"`. `NULL` (default) tries 0.51, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8 and 0.9 for `"majority"`, and 0.1 to 0.8 by steps of 0.1 for `"jaccard"`.
`similarity_types`	Subset of `c("complete", "partial")`: the types to try (see `merge_dynamic_clusters()`).

`min_length`	Whole number, the number of windows from which a dynamic cluster is called *persistent* (3 by default).
`verbose`	Logical. If `TRUE` (default), reports progress.

Details

The matching is run for each setting (threshold and similarity type) with `merge_dynamic_clusters()`. For each setting, with a *window cluster* meaning a cluster of one window and a *dynamic cluster* the set of window clusters that share an identifier, the columns of the result are:

- `n_window_clusters`: the number of window clusters (the same for all settings);
- `n_dynamic_clusters`: the number of dynamic clusters;
- `share_continued`: the share of the window clusters of the second window onwards that inherit the identifier of a cluster of the previous window (high: many continuities);
- `median_lifespan`, `max_lifespan`: the number of windows in which a dynamic cluster exists (median and maximum over dynamic clusters);
- `share_single_window`: the share of dynamic clusters that exist in one window only;
- `n_persistent`: the number of dynamic clusters existing in at least `min_length` windows;
- `share_nodes_persistent`: the share of the nodes of all windows (a node present in two windows counts twice) that belong to persistent dynamic clusters;
- `largest_share`: the share of those nodes in the largest dynamic cluster.

The table of the lifespans of each dynamic cluster is stored in the attribute `"lifespans"` and used by `plot_merge_diagnostics()`.

A good threshold is on a *plateau*: the main dynamic clusters, and the share of nodes in persistent clusters, change little around it. Windows without any shared node cannot be matched whatever the threshold, and are reported in a warning.

Value

A data frame with one row per threshold and similarity type, the `method` and the columns described above, with the table of lifespans in its `"lifespans"` attribute.

See Also

`merge_dynamic_clusters()`, `plot_merge_diagnostics()`.

Examples

```
networks <- build_dynamic_networks(
  nodes = subset(Nodes_stagflation, source_type == "Stagflation"),
  directed_edges = Ref_stagflation,
  source_id = "source_id",
  target_id = "target_id",
  time_variable = "source_year",
  time_window = 10,
  window_step = 4,
  method = "angle",
```

```
  edges_threshold = 2,
  verbose = FALSE
)
networks <- add_clusters(
  networks,
  clustering_method = "leiden",
  objective_function = "modularity",
  seed = 123,
  verbose = FALSE
)

diagnostics <- merge_diagnostics(
  networks,
  cluster_id = "cluster_leiden",
  node_id = "source_id",
  method = "majority",
  thresholds = c(0.51, 0.6, 0.75),
  similarity_types = "partial",
  verbose = FALSE
)
diagnostics[, c("similarity_type", "threshold", "n_dynamic_clusters", "median_lifespan")]
```

`merge_dynamic_clusters` *Follow clusters across time windows*

Description

[Experimental]

When clusters are detected in each time window separately, their identifiers mean nothing from one window to the next: cluster "01" of the 1975-1984 network and cluster "01" of the 1976-1985 network may be two different literatures. `merge_dynamic_clusters()` links the clusters of successive windows that share most of their nodes, and gives them a common identifier, so that a cluster becomes an entity that can be born, persist and disappear. This is the step that turns a series of networks into a history of research communities, in the spirit of the alluvial maps of Rosvall and Bergstrom (2010).

Two rules are available, chosen with `method`. Let V_a be the nodes of cluster a in window $t-1$ and V_b those of cluster b in window t.

- `method = "majority"`, a **double majority**. Cluster b inherits the identifier of a when most nodes of a go to b *and* most nodes of b come from a:

$$\frac{|V_a \cap V_b|}{D_a} > \tau \quad \text{and} \quad \frac{|V_a \cap V_b|}{D_b} > \tau,$$

  with τ the `threshold_similarity`, strictly above 0.5. Since $\tau > 0.5$, a cluster can inherit at most one identifier, and an identifier passes to at most one cluster.

- `method = "jaccard"`, an **overlap matching with a Jaccard threshold**, as in the tracking of dynamic communities of Greene et al. (2010). The Jaccard index of two clusters is the share of their union that they have in common,

$$J(a,b) = \frac{|V_a \cap V_b|}{|V_a \cup V_b|}.$$

  The pairs with $J(a,b) \geq \theta$, θ being the `threshold_similarity`, are candidate matches. They are taken one by one, by decreasing J (then by decreasing number of shared nodes), skipping the pairs where a or b has already been matched: each past identifier goes to the present cluster that resembles it most, and each cluster inherits at most one identifier. θ can be below 0.5: a cluster that loses a large part of its members to another cluster, or that doubles with newcomers, can keep its identity if it stays the best match of its predecessor.

In both rules, a cluster that meets no match gets a new identifier. The sets V depend on `similarity_type`: with `"complete"`, they are all the nodes of the clusters ($D_a = |V_a|$, $D_b = |V_b|$); with `"partial"`, only the nodes present in both windows count ($D_a = |V_a \cap V_t|$, $D_b = |V_b \cap V_{t-1}|$, and the union of the Jaccard index is restricted to them), V_t being all the nodes of window t.

There is no default `method`, `threshold_similarity` or `similarity_type`: they change the dynamic clusters, so the choice must be explicit, and `verbose = TRUE` reports the rule that is applied.

Usage

```
merge_dynamic_clusters(
  graphs,
  cluster_id,
  node_id,
  method = c("majority", "jaccard"),
  threshold_similarity,
  similarity_type = c("complete", "partial"),
  verbose = TRUE
)
```

Arguments

`graphs`	A list of at least two `tbl_graphs` in chronological order (oldest first), as produced by `build_dynamic_networks()` and `add_clusters()`.
`cluster_id`	Character scalar, the node column with the clusters of each window (e.g. `"cluster_leiden"`).
`node_id`	Character scalar, the node column with an identifier that is stable across windows (e.g. `"source_id"`).
`method`	`"majority"` (double majority) or `"jaccard"` (overlap matching with a Jaccard threshold), see the description. Required.
`threshold_similarity`	The threshold of the rule. For `method = "majority"`, τ in $(0.5, 1]$: a cluster is linked when the share is *above* it. For `method = "jaccard"`, θ in $(0, 1]$: two clusters can be linked when their Jaccard index is *at least* it. Higher values require more similar clusters and create more distinct dynamic clusters. Required.
`similarity_type`	`"complete"` or `"partial"`, see the description. Required.
`verbose`	Logical. If `TRUE` (default), says which rule is applied.

Details

Which windows? The rules need nodes present in successive windows. In a network of documents (e.g. bibliographic coupling), each document has a single date, so windows that do not overlap (`window_step` equal to `time_window`, the default of `build_dynamic_networks()`) share no node and no cluster can be matched: the windows must overlap (`window_step` smaller than `time_window`). Non-overlapping windows work when nodes persist over time, as cited references (co-citation), authors or journals do; two windows may still share no node when the literature changed completely.

Complete or partial? With overlapping windows, the network of window t loses the documents of the first years of window $t-1$ and gains those of new years; in a growing literature, newcomers can be numerous. With `"complete"`, a cluster that keeps all its past members but doubles with newcomers cannot inherit the identifier under the double majority, because its members coming from a are less than half of it: the cluster is considered new because its composition changed. With `"partial"`, the newcomers are ignored and the same cluster keeps its identifier. Which one is right depends on what a cluster is taken to be: a group of documents (complete), or a structure that recruits new documents (partial).

Majority or Jaccard? The double majority is demanding and transparent: two clusters are the same if each is mostly made of the other, and the identifier is never ambiguous. The Jaccard matching is more flexible: with a low θ, it follows a cluster that splits (the largest piece, the best match, keeps the identifier) or that merges with another. Both use the nodes of consecutive windows only; compare them with `merge_diagnostics()`.

What the rules do not do. Clusters are compared only with the previous window: a cluster that vanishes for one window and comes back gets a new identifier. Splits and merges are not recorded as such: one part keeps the identifier, the others are new. With a threshold close to its minimum, small fluctuations of the partition (a stochastic algorithm, a resolution change) may break or create continuities; it is good practice to compare the results for a few thresholds and seeds.

Value

The list of graphs, with the same names, where nodes have a new column `dynamic_{cluster_id}` (e.g. `dynamic_cluster_leiden`, with identifiers "cl_1", "cl_2"...) and edges the columns `dynamic_{cluster_id}_from`, `dynamic_{cluster_id}_to` and `dynamic_{cluster_id}` (missing for edges between clusters).

References

Greene D, Doyle D, Cunningham P (2010). "Tracking the Evolution of Communities in Dynamic Social Networks." In *2010 International Conference on Advances in Social Networks Analysis and Mining*, 176–183. doi:10.1109/ASONAM.2010.17.

Rosvall M, Bergstrom CT (2010). "Mapping Change in Large Networks." *PLoS ONE*, **5**(1), e8694. doi:10.1371/journal.pone.0008694.

See Also

`name_clusters()` with `name_dynamic_clusters = TRUE` to label the dynamic clusters; `networks_to_alluv()` and `plot_alluvial()` to draw their evolution.

Examples

```
networks <- build_dynamic_networks(
  nodes = subset(Nodes_stagflation, source_type == "Stagflation"),
  directed_edges = Ref_stagflation,
  source_id = "source_id",
  target_id = "target_id",
  time_variable = "source_year",
  time_window = 10,
  method = "similarity",
  edges_threshold = 1,
  verbose = FALSE
)
networks <- add_clusters(
  networks,
  clustering_method = "leiden",
  objective_function = "modularity",
  seed = 123,
  verbose = FALSE
)

networks <- merge_dynamic_clusters(
  networks,
  cluster_id = "cluster_leiden",
  node_id = "source_id",
  method = "majority",
  threshold_similarity = 0.51,
  similarity_type = "partial"
)
networks[[2]]

# The same windows, with the Jaccard matching
merge_dynamic_clusters(
  networks,
  cluster_id = "cluster_leiden",
  node_id = "source_id",
  method = "jaccard",
  threshold_similarity = 0.3,
  similarity_type = "partial"
)
```

---

minimize_crossing_alluvial

Order the clusters of an alluvial to reduce crossing flows

Description

[Experimental]

In an alluvial plot, flows between clusters drawn far apart cross many other flows and make the plot unreadable. `minimize_crossing_alluvial()` computes an order of the clusters that places

clusters exchanging many nodes next to each other. It is a heuristic, not an optimal solution to the crossing problem.

The order is built in two steps. First, the function builds a network of clusters: two clusters are tied when nodes move from one to the other between windows, with a cosine weight $n_{ab}/\sqrt{\bar{s}_a \bar{s}_b}$, n_{ab} being the number of moving nodes and $\bar{s}_a$ the mean size of cluster a over its windows. It removes the ties one by one, from the weakest to the strongest, and records at each step the connected component of each cluster. Clusters are ordered by this sequence of components: clusters that stay together until the strongest ties are removed end up side by side (this is the order of the leaves of a single-linkage hierarchical clustering of the clusters, computed with `stats::hclust()`). This groups clusters that exchange nodes, but it does not count the crossings.

Second (`refine = TRUE`, the default), the order is improved against the crossings themselves. For each pair of consecutive windows, the flows are the numbers of nodes going from a cluster of the first window to a cluster of the second; two flows cross when the order of their origins and the order of their destinations are opposite, and the pair is weighted by the product of the two flows. Each cluster in turn is moved to the position that minimises the total, until no move helps. This is a local search, which can end in a local optimum, and which costs more with many clusters and flows. For large alluvials (more than 100 clusters, or more than two million pairs of flows), the barycentre heuristic is used instead: each cluster is placed at the mean position of the clusters it exchanges nodes with, as long as this reduces the crossings. Filtering the small clusters first (see `networks_to_alluv()`) gives faster and more readable alluvials.

Usage

```
minimize_crossing_alluvial(
  alluv_dt,
  cluster_id = NULL,
  node_id = NULL,
  window_column = "window",
  refine = TRUE,
  max_passes = 10,
  verbose = TRUE
)
```

Arguments

`alluv_dt`	A data frame created with `networks_to_alluv()`, whose rows follow the chronological order of the windows (the order of the levels of `window_column` if it is a factor, as in `networks_to_alluv()`, or of their appearance).
`cluster_id`	Character scalar, the column with the dynamic clusters.
`node_id`	Character scalar, the column with the node identifiers.
`window_column`	Character scalar, the column with the time windows (`"window"` by default).
`refine`	Logical. If `TRUE` (default), refines the order against the crossings (see the description); if `FALSE`, keeps the order of the components.
`max_passes`	Whole number, the maximum number of passes of the refinement over all the clusters.
`verbose`	Logical. If `TRUE` (default), reports the weighted number of crossings before and after the refinement.

Details

The alluvial method (following clusters over time with `networks_to_alluv()`, `minimize_crossing_alluvial()` and `plot_alluvial()`) was initiated by Alexandre Truc.

Value

`alluv_dt` with a new column `minimize_crossing_order`, the rank of each cluster, that `plot_alluvial()` uses to order the blocks.

Examples

```
networks <- build_dynamic_networks(
  nodes = subset(Nodes_stagflation, source_type == "Stagflation"),
  directed_edges = Ref_stagflation,
  source_id = "source_id",
  target_id = "target_id",
  time_variable = "source_year",
  time_window = 10,
  method = "similarity",
  edges_threshold = 1,
  verbose = FALSE
)
networks <- add_clusters(
  networks,
  clustering_method = "leiden",
  objective_function = "modularity",
  seed = 123,
  verbose = FALSE
)
networks <- merge_dynamic_clusters(
  networks,
  cluster_id = "cluster_leiden",
  node_id = "source_id",
  method = "majority", threshold_similarity = 0.51,
  similarity_type = "partial"
)
alluv_dt <- networks_to_alluv(
  networks,
  cluster_id = "dynamic_cluster_leiden",
  node_id = "source_id"
)

alluv_dt <- minimize_crossing_alluvial(
  alluv_dt,
  cluster_id = "dynamic_cluster_leiden",
  node_id = "source_id"
)
unique(alluv_dt[, c("dynamic_cluster_leiden", "minimize_crossing_order")])
```

name_clusters *Give readable names to clusters*

Description

[Experimental]

A cluster identifier ("01", "cl_12") says nothing about its content. `name_clusters()` gives each cluster a provisional label that helps reading plots and tables, with one of four methods: the words that distinguish the cluster from the others (`"tf-idf"`), the metadata of its most central node (`"tidygraph_functions"`) or of its node with the highest value of a given column (`"given_column"`), or a label written by a language model from a description of the cluster (`"llm"`, optional, through the `ellmer` package: a model on your computer or an online service). These labels are starting points for interpretation, not the result of it: naming a cluster requires reading its documents, the references they share, and what distinguishes them from neighbouring clusters (Ding 2011).

Usage

```
name_clusters(
  graphs,
  method = c("tidygraph_functions", "given_column", "tf-idf", "llm"),
  name_dynamic_clusters = FALSE,
  cluster_id,
  label_columns,
  label_name = "cluster_label",
  tidygraph_function = NULL,
  order_by = NULL,
  text_columns = NULL,
  nb_terms_label = 3,
  prompt_columns = NULL,
  nb_nodes = NULL,
  llm = NULL,
  instructions = NULL,
  context = NULL,
  preview = FALSE,
  name_merged_clusters = NULL,
  ...
)
```

Arguments

`graphs`	A `tbl_graph` or a list of `tbl_graphs`.
`method`	`"tf-idf"` (chosen when no method is given), `"tidygraph_functions"`, `"given_column"` or `"llm"` (see the details).
`name_dynamic_clusters`	Logical. `TRUE` when `cluster_id` identifies clusters across a list of graphs (the dynamic clusters of `merge_dynamic_clusters()`): each cluster then gets one

name, computed on all its nodes in all windows. FALSE (default) names the clusters of each graph separately.

cluster_id
: Character scalar, the node column with the clusters to name.

label_columns
: For "tidygraph_functions" and "given_column", the node column(s) used to build the label of the chosen node, joined by "_" (e.g. c("source_author", "source_year")).

label_name
: Character scalar, the name of the new column ("cluster_label" by default).

tidygraph_function
: For "tidygraph_functions", a call to a centrality function of tidygraph, such as tidygraph::centrality_pagerank() (see tidygraph::centrality).

order_by
: Character scalar, a numeric node column used to rank the nodes of each cluster, for two methods (ignored by the others):

- "given_column": the node with the highest value names the cluster, e.g. "node_size" from build_network() with compute_size = TRUE (the most cited document of the cluster);
- "llm" (required): the nb_nodes nodes with the highest values are shown to the model, e.g. "within_module_degree" from add_node_roles() (the nodes most tied to their own cluster). For dynamic clusters, a node present in several windows is ranked by its highest value.

text_columns
: For "tf-idf", the node column(s) with text (titles, abstracts, keywords), passed to extract_tfidf().

nb_terms_label
: For "tf-idf", the number of terms in each label (3 by default), separated by commas. A term that is part of a term already in the label, or contains one, is skipped ("business" after "business cycle").

prompt_columns
: For "llm" (required), character vector, the node column(s) written in the description of each cluster sent to the model: what the model knows about the nodes (e.g. c("source_title", "source_year", "cluster_label")). A column with one value per cluster (a label computed before, e.g. with method = "tf-idf") is given once for the cluster.

nb_nodes
: For "llm" (required), whole number, the number of nodes listed for each cluster, highest order_by first. 0 lists none: the model then reads only the context and the columns with one value per cluster.

llm
: For "llm", the model: a chat object of ellmer, configured as you would for any other use, e.g. ellmer::chat_ollama(model = "mistral", params = ellmer::params(temperature = 0, seed = 123)) (see the details). Not needed with preview = TRUE.

instructions
: For "llm", character scalar, the system prompt sent before the description of each cluster. NULL (default) uses the instructions given in the details. Change them to ask for longer labels, another language, another vocabulary... To start from the default, get it with preview = TRUE (column instructions), e.g. instructions = paste(prompts$instructions[1], "Answer in French.").

context
: For "llm", optional character scalar describing the network to the model: what the nodes are, the field, the period, the question (e.g. "Economics articles on stagflation, 1970-2010, linked by their shared references"), written at the top of each description. Strongly recommended: in our tests on three corpora and three local models, it raised the quality of the labels more than any other change to the description (see the details).

`preview` For `"llm"`, logical. If `TRUE`, returns what the model would receive, without calling it (no model needed): a tibble with one row per cluster, its identifier, the `instructions` (the system prompt) and the `description` of the cluster (the user message), plus a `graph` column for a list of graphs named separately. Print a description with `cat()`. `FALSE` (default) names the clusters.

`name_merged_clusters`
[Deprecated] Former name of `name_dynamic_clusters`.

`...` For `"tf-idf"`, other arguments of `extract_tfidf()` (e.g. `clean_word_method`, `n_gram`, `ngrams_filter`). Must be empty for the other methods (a misspelt argument is an error).

Details

tf-idf. Each cluster is treated as one document made of the texts of its nodes, and each term t (word or n-gram) gets the weight

$$\mathrm{tf}_{tc} \times \mathrm{idf}_t = \frac{n_{tc}}{\sum_u n_{uc}} \times \log \frac{C}{\mathrm{df}_t}$$

(Spärck Jones 1972; Salton and Buckley 1988), with n_{tc} the number of occurrences of t in cluster c, C the number of clusters and df_t the number of clusters where t appears. The label gathers the terms with the highest weight: frequent in the cluster, rare in the others. With `name_dynamic_clusters = FALSE`, the comparison is made within each graph, so clusters of different windows may get the same label.

Most central node. `"tidygraph_functions"` computes the centrality given in `tidygraph_function` on each graph (it is added as a node column) and labels each cluster with the node with the highest value. The centrality is computed on the whole graph, not within the cluster. This method needs a network structure and therefore cannot be used with `name_dynamic_clusters = TRUE`.

Given column. `"given_column"` labels each cluster with its node with the highest `order_by` value: for an article network, typically its most cited article.

Language model. `"llm"` writes a description of each cluster and asks the model given in `llm` for a label. The model is a chat object of the `ellmer` package (only needed for this method), so any model ellmer reaches can be used:

- a model running on your computer, so that the content of the nodes stays on it: `ellmer::chat_ollama()` (Ollama), `chat_lmstudio()`, `chat_vllm()`, or `chat_openai_compatible()` for another local server (e.g. llama.cpp);
- an online service (`chat_openai()`, `chat_anthropic()`...): the descriptions of the clusters are then sent to it.

Configure the chat object as for any other use of ellmer; for more stable labels, set `params = ellmer::params(temperature = 0, seed = 123)` (this does not guarantee the same labels on another computer, and some online services ignore `seed`). Each cluster is sent alone to a copy of `llm` (its earlier conversation is not sent), with `instructions` as the system prompt, and the answer is constrained to one label (`chat_structured()`). The label is cleaned: no quotes or final period, words separated by spaces ("Asset_Pricing" becomes "Asset Pricing"). The default instructions are:

`` `r llm_default_instructions` ``

Writing the prompt. We compared nine prompts, with three local models (`gemma3:4b`, `gemma3:12b`, `qwen3:14b`) on 21 clusters of three corpora (articles of economics), the labels being scored blind by another model (accuracy and specificity, from 1 to 5; a small test, to be taken as an indication). What mattered most, in order:

1. The instructions. Short ones ("a label of 2 or 3 words") give generic labels ("Monetary policy analysis"); the default above, which asks for the specific topic, forbids empty words and gives examples, raised the scores by 0.5 to 0.8 points whatever the model.
2. The `context`: one sentence on what the nodes are, the field and the period raised the scores by a further 0.25 to 0.5 points. It helps the model most where it cannot tell the cluster apart by itself.
3. Little else: 10 nodes are enough (25 did not help, 5 was a bit worse); year and journal in `prompt_columns` changed little; and a column of key terms (tf-idf) did not help and produced more repeated labels. A larger model did not systematically beat a small one once the prompt was good. If you change `instructions`, keep the rules asking for a specific topic and keep the `context`.

With `ellmer::chat_ollama()`, a model that thinks before answering (e.g. `qwen3`) can take minutes per cluster or time out, because ellmer reaches Ollama through its OpenAI-compatible endpoint, where `think = FALSE` is ignored: use `api_args = list(reasoning_effort = "none")`.

The description of a cluster gathers, when available:

- the `context` you give;
- the number of nodes of the cluster (and of windows, for dynamic clusters);
- the columns of `prompt_columns` with one value per cluster (e.g. a tf-idf label computed before with `method = "tf-idf"`), given once;
- its `nb_nodes` main nodes (highest `order_by`), described by the other columns of `prompt_columns`.

Everything comes from the node tables of `graphs`. The value of a node longer than 300 characters (an abstract) is cut, and its line breaks are removed; columns with one value per cluster are written in full. Use `preview = TRUE` to read the descriptions before running the model.

If the model fails on the first cluster, `name_clusters()` stops: the problem is likely its configuration (try `llm$chat("Hello")`). A later cluster whose label fails is labelled "no_name" with a warning. By default, each answer may take up to 300 seconds, with up to 3 tries (the options `ellmer_timeout_s` and `ellmer_max_tries` of ellmer): raise `options(ellmer_timeout_s = )` for a slow model. A warning is given when a description exceeds about 4000 tokens: some local servers cut longer prompts without warning (Ollama has a context of 4096 tokens by default).

Nodes of an unnamed cluster get "no_name".

Value

The same object as `graphs`, with a new node column `label_name`, also joined to the edges through their cluster column. With `method = "llm"` and `preview = TRUE`, a tibble of the messages that would be sent to the model (see `preview`).

References

Ding Y (2011). "Community Detection: Topological vs. Topical." *Journal of Informetrics*, **5**(4), 498–514. doi:10.1016/j.joi.2011.02.006.

Salton G, Buckley C (1988). "Term-Weighting Approaches in Automatic Text Retrieval." *Information Processing & Management*, **24**(5), 513–523.

Spärck Jones K (1972). "A Statistical Interpretation of Term Specificity and Its Application in Retrieval." *Journal of Documentation*, **28**(1), 11–21.

See Also

`extract_tfidf()` to see more terms than the labels show.

Examples

```
network <- build_network(
  nodes = subset(Nodes_stagflation, source_type == "Stagflation"),
  directed_edges = Ref_stagflation,
  source_id = "source_id",
  target_id = "target_id",
  method = "similarity",
  edges_threshold = 1
)
network <- add_clusters(
  network,
  clustering_method = "leiden",
  objective_function = "modularity",
  seed = 123,
  verbose = FALSE
)

# The words of the titles that distinguish each cluster
network <- name_clusters(
  network,
  method = "tf-idf",
  cluster_id = "cluster_leiden",
  text_columns = "source_title",
  nb_terms_label = 3,
  clean_word_method = "none"
)

# The article with the highest PageRank of each cluster
network <- name_clusters(
  network,
  method = "tidygraph_functions",
  cluster_id = "cluster_leiden",
  label_columns = c("source_author", "source_year"),
  label_name = "central_article",
  tidygraph_function = tidygraph::centrality_pagerank()
)
```

```
unique(tidygraph::as_tibble(network, active = "nodes")[
  c("cluster_leiden", "cluster_label", "central_article")
])

# What a language model would read for each cluster (no model needed):
# titles, years and the tf-idf label, for the 10 nodes most tied to their
# own cluster
network <- add_node_roles(network, cluster_id = "cluster_leiden")
prompts <- name_clusters(
  network,
  method = "llm",
  cluster_id = "cluster_leiden",
  prompt_columns = c("source_title", "source_year", "cluster_label"),
  order_by = "within_module_degree",
  nb_nodes = 10,
  context = "Economics articles on stagflation, linked by their shared references.",
  preview = TRUE
)
cat(prompts$instructions[1])
cat(prompts$description[1])

## Not run:
# A label written by a model running on your computer, here with Ollama
# (install Ollama, then `ollama pull mistral` in a terminal)
network <- name_clusters(
  network,
  method = "llm",
  cluster_id = "cluster_leiden",
  prompt_columns = c("source_title", "source_year", "cluster_label"),
  order_by = "within_module_degree",
  nb_nodes = 10,
  label_name = "llm_label",
  llm = ellmer::chat_ollama(model = "mistral",
                            params = ellmer::params(temperature = 0, seed = 123)),
  context = "Economics articles on stagflation, linked by their shared references."
)

## End(Not run)
```

`networks_to_alluv` *Turn a list of networks into the table of an alluvial plot*

Description

[Experimental]

An alluvial plot shows the evolution of dynamic clusters: each time window is a column, each cluster a block whose height is its share of the nodes of the window, and the flows between blocks follow the nodes that stay in the network from one window to the next (Rosvall and Bergstrom 2010). `networks_to_alluv()` stacks the node tables of a list of networks into the long table that such a plot needs (one row per node and window) and adds summary statistics on the clusters.

Usage

```
networks_to_alluv(
  graphs,
  cluster_id,
  node_id,
  summary_cluster_stats = TRUE,
  keep_color = TRUE,
  color_column = "color",
  keep_cluster_label = TRUE,
  cluster_label_column = "cluster_label"
)
```

Arguments

graphs	A list of tbl_graphs with dynamic clusters (see merge_dynamic_clusters()).
cluster_id	Character scalar, the node column with the dynamic clusters (e.g. "dynamic_cluster_leiden").
node_id	Character scalar, the node column with a stable identifier; it is the "alluvium" that the flows follow.
summary_cluster_stats	Logical. If TRUE (default), adds the statistics described in the return value, useful to keep only the main clusters in the plot.
keep_color	Logical. If TRUE (default), keeps color_column.
color_column	Character scalar, the colour column ("color", from color_networks()).
keep_cluster_label	Logical. If TRUE (default), keeps cluster_label_column.
cluster_label_column	Character scalar, the label column ("cluster_label", from name_clusters()).

Details

The alluvial method (following clusters over time with networks_to_alluv(), minimize_crossing_alluvial() and plot_alluvial()) was initiated by Alexandre Truc.

Value

A data.table with one row per node and window: window (the name of the graph in the list), node_id, the dynamic cluster, the colour and label columns if kept, y_alluv ($1/n_t$, n_t being the number of nodes of the window, so that each window has a total height of 1), and, with summary_cluster_stats = TRUE:

- share_cluster_alluv: the share of all rows that belong to the cluster, in per cent;
- share_cluster_window: the share of the window's nodes in the cluster, in per cent;
- share_cluster_max: the maximum of share_cluster_window over the windows;
- length_cluster: the number of windows where the cluster exists.

References

Rosvall M, Bergstrom CT (2010). "Mapping Change in Large Networks." *PLoS ONE*, **5**(1), e8694. doi:10.1371/journal.pone.0008694.

See Also

`minimize_crossing_alluvial()`, `prepare_label_alluvial()`, `color_alluvial()` and `plot_alluvial()`.

Examples

```
networks <- build_dynamic_networks(
  nodes = subset(Nodes_stagflation, source_type == "Stagflation"),
  directed_edges = Ref_stagflation,
  source_id = "source_id",
  target_id = "target_id",
  time_variable = "source_year",
  time_window = 10,
  method = "similarity",
  edges_threshold = 1,
  verbose = FALSE
)
networks <- add_clusters(
  networks,
  clustering_method = "leiden",
  objective_function = "modularity",
  seed = 123,
  verbose = FALSE
)
networks <- merge_dynamic_clusters(
  networks,
  cluster_id = "cluster_leiden",
  node_id = "source_id",
  method = "majority", threshold_similarity = 0.51,
  similarity_type = "partial"
)

alluv_dt <- networks_to_alluv(
  networks,
  cluster_id = "dynamic_cluster_leiden",
  node_id = "source_id"
)
alluv_dt[1:5]
```

`Nodes_coupling` *Nodes of a ready-made bibliographic coupling network on the stagflation*

Description

The 154 documents of Nodes_stagflation that explain the stagflation and have references in Ref_stagflation. With Edges_coupling, it gives a bibliographic coupling network already computed, for instance to try the functions of the package on a `tbl_graph` built directly with `tidygraph::tbl_graph()`.

Usage

```
Nodes_coupling
```

Format

A data frame with 154 rows and 6 variables:

source_id Identifier of the document, as a character string.

source_author First author of the document.

source_label Short label (author and year), for plots.

source_year Year of publication.

source_title Title of the document.

source_journal Journal of publication, for articles.

Source

Built from Nodes_stagflation and Ref_stagflation by `inst/data-raw/creating_network_data.R`.

`Nodes_stagflation` *Documents explaining the 1970s US stagflation, and the documents they cite*

Description

The corpus of Goutsmedt (2021): the 174 books and academic articles that tried to explain the US stagflation of the 1970s, published from 1975 to 2013 (`source_type == "Stagflation"`), and the 480 documents cited at least twice by them (`"Non-Stagflation"`). With Ref_stagflation, it is the example data of the package: the explanations are the nodes of a bibliographic coupling network, linked by the references they share.

Usage

```
Nodes_stagflation
```

Format

A data frame with 654 rows and 7 variables:

source_id Identifier of the document.

source_author First author of the document (see Authors_stagflation for all authors).

source_label Short label (author and year), for plots.

source_year Year of publication, as a character string.

source_title Title of the document.

source_journal Journal of publication, for articles.

source_type `"Stagflation"` for the documents explaining the stagflation, `"Non-Stagflation"` for the documents they cite.

Source

Goutsmedt (2021).

References

Goutsmedt A (2021). "From the Stagflation to the Great Inflation: Explaining the US Economy of the 1970s." *Revue d'Économie Politique*, **131**(3), 557–582. https://www.cairn.info/revue-d-economie-politique-2021-3-page-557.htm.

`plot_alluvial` *Plot the evolution of dynamic clusters as an alluvial*

Description

[Experimental]

`plot_alluvial()` draws the table created by `networks_to_alluv()` with ggalluvial: one column per time window, one block per dynamic cluster, with a height proportional to its share of the window, and flows that follow the nodes from one window to the next. It can colour the clusters (`color_alluvial()`), reorder them to limit crossings (`minimize_crossing_alluvial()`) and position their labels (`prepare_label_alluvial()`) on the way.

Usage

```
plot_alluvial(
  alluv_dt,
  cluster_id,
  node_id,
  window_column = "window",
  color_column = "color",
  color_alluvial = FALSE,
  color = NULL,
  minimize_crossing = FALSE,
```

```
  prepare_label = FALSE,
  cluster_label_column = "cluster_label",
  min_share_cluster = NULL,
  min_windows_cluster = NULL,
  print_plot_code = FALSE
)
```

Arguments

alluv_dt	A data frame created with networks_to_alluv(), or directly the list of tbl_graphs with dynamic clusters (see merge_dynamic_clusters()), which is then converted with networks_to_alluv() (color_column and cluster_label_column are the columns kept).
cluster_id	Character scalar, the column with the dynamic clusters (e.g. "dynamic_cluster_leiden").
node_id	Character scalar, the column with the node identifiers.
window_column	Character scalar, the column with the time windows ("window" by default).
color_column	Character scalar, the column with the colours of the clusters ("color" by default). If absent or NULL, ggplot2 chooses the colours.
color_alluvial	Logical. If TRUE, runs color_alluvial() first.
color	With color_alluvial = TRUE, the colours passed to color_alluvial().
minimize_crossing	Logical. If TRUE, runs minimize_crossing_alluvial() first. Clusters are also reordered when alluv_dt already has a minimize_crossing_order column.
prepare_label	Logical. If TRUE, runs prepare_label_alluvial() first, which adds a label_x column with one label per cluster.
cluster_label_column	Character scalar, the column with the labels used by prepare_label_alluvial() ("cluster_label" by default).
min_share_cluster, min_windows_cluster	Optional numbers to keep the main clusters only: the clusters whose largest share of a window (share_cluster_max, in per cent) is at least min_share_cluster, and which exist in at least min_windows_cluster windows (length_cluster). NULL (default) keeps all the clusters.
print_plot_code	Logical. If TRUE, prints the ggplot2 code of the plot.

Details

Labels are drawn with ggrepel, which must be installed, when alluv_dt has a label_x column. Small clusters clutter alluvials: filtering the table on share_cluster_max or length_cluster (see networks_to_alluv()) before plotting keeps the main ones.

The alluvial method was initiated by Alexandre Truc.

Value

A ggplot object.

Examples

```
networks <- build_dynamic_networks(
  nodes = subset(Nodes_stagflation, source_type == "Stagflation"),
  directed_edges = Ref_stagflation,
  source_id = "source_id",
  target_id = "target_id",
  time_variable = "source_year",
  time_window = 10,
  method = "similarity",
  edges_threshold = 1,
  verbose = FALSE
)
networks <- add_clusters(
  networks,
  clustering_method = "leiden",
  objective_function = "modularity",
  seed = 123,
  verbose = FALSE
)
networks <- merge_dynamic_clusters(
  networks,
  cluster_id = "cluster_leiden",
  node_id = "source_id",
  method = "majority", threshold_similarity = 0.51,
  similarity_type = "partial"
)
alluv_dt <- networks_to_alluv(
  networks,
  cluster_id = "dynamic_cluster_leiden",
  node_id = "source_id"
)
if (requireNamespace("ggalluvial", quietly = TRUE)) {
  plot_alluvial(
    alluv_dt,
    cluster_id = "dynamic_cluster_leiden",
    node_id = "source_id",
    color_alluvial = TRUE,
    minimize_crossing = TRUE
  )
}
```

---

plot_merge_diagnostics

Plot the diagnostics of the matching of clusters across windows

---

Description

[Experimental]

Draws the result of `merge_diagnostics()`. Two views are available:

- `"summary"` (default): one panel per indicator (number of dynamic clusters, share of window clusters that find a predecessor, median lifespan, share of nodes in persistent clusters...), against the threshold, with one line per type of similarity. Look for plateaus: thresholds around which the indicators barely move.
- `"lifespans"`: for each threshold, the share of the nodes (of all windows) that belong to dynamic clusters of a given lifespan (1 window, 2, 3 to 5, 6 to 10, more than 10), one panel per type of similarity. It shows how the weight of short-lived and long-lived clusters changes with the threshold.

Usage

```
plot_merge_diagnostics(
  diagnostics,
  view = c("summary", "lifespans"),
  metrics = c("n_dynamic_clusters", "share_continued", "median_lifespan",
    "share_nodes_persistent"),
  base_size = 16
)
```

Arguments

`diagnostics`	The result of `merge_diagnostics()`.
`view`	`"summary"` or `"lifespans"`.
`metrics`	For the summary, the columns of `diagnostics` to draw.
`base_size`	Base font size of the plot.

Value

A ggplot object.

See Also

`merge_diagnostics()`.

Examples

```
networks <- build_dynamic_networks(
  nodes = subset(Nodes_stagflation, source_type == "Stagflation"),
  directed_edges = Ref_stagflation,
  source_id = "source_id",
  target_id = "target_id",
  time_variable = "source_year",
  time_window = 10,
  window_step = 4,
  method = "angle",
  edges_threshold = 2,
  verbose = FALSE
)
```

```
networks <- add_clusters(
  networks,
  clustering_method = "leiden",
  objective_function = "modularity",
  seed = 123,
  verbose = FALSE
)
diagnostics <- merge_diagnostics(
  networks, "cluster_leiden", "source_id", method = "majority",
  thresholds = c(0.51, 0.6, 0.75), similarity_types = "partial", verbose = FALSE
)
plot_merge_diagnostics(diagnostics)
plot_merge_diagnostics(diagnostics, view = "lifespans")
```

`plot_networks` *Plot networks*

Description

[Experimental]

`plot_networks()` draws a network from the columns created by the previous steps of the pipeline: coordinates (`layout_networks()`), colours (`color_networks()`) and cluster labels (`name_clusters()` and `prepare_label_networks()`). Edges are drawn as light arcs whose width follows their weight, nodes as points whose size follows `node_size_column`, and each cluster gets a label at its centre of gravity.

It is meant as a quick, ready-made figure. For a figure to publish, `print_plot_code = TRUE` prints the `ggraph` code, which can be copied and adapted.

Usage

```
plot_networks(
  graphs,
  x = "x",
  y = "y",
  cluster_label_column = "cluster_label",
  node_size_column = "node_size",
  color_column = "color",
  color_networks = FALSE,
  color = NULL,
  max_arcs = 20000,
  print_plot_code = FALSE
)
```

Arguments

`graphs`	A `tbl_graph` or a list of `tbl_graphs`.

x, y — Character scalars, the node columns with the coordinates ("x" and "y" by default).

cluster_label_column — Character scalar, the node column with the labels to display: "cluster_label" (name_clusters()) by default, or a cluster column such as cluster_leiden.

node_size_column — Character scalar, the numeric node column giving the size of nodes: "node_size" by default (see compute_size in build_network()). If NULL or absent, all nodes have the same size.

color_column — Character scalar, the column with the colours of nodes and edges ("color" by default, from color_networks()).

color_networks — Logical. If TRUE, color_networks() is run first, with cluster_label_column as the categories.

color — With color_networks = TRUE, the colours passed to color_networks().

max_arcs — Whole number. Edges are drawn as arcs whose width follows their weight, unless a graph has more than max_arcs edges (20,000 by default): they are then drawn as straight lines of constant width, which is much faster (on a network of 140,000 edges, about 2 seconds and 1 GB of memory to draw, against 100 seconds and 11 GB with arcs). Set it to Inf to always draw arcs.

print_plot_code — Logical. If TRUE, prints the ggplot2/ggraph code of the plot.

Details

Missing pieces are filled in: labels are positioned with prepare_label_networks() when label_x is absent, and edges without a weight column get a weight of 1. Labels are drawn with ggrepel, which must be installed.

Value

For a tbl_graph, a ggplot object. For a list of graphs, a named list of ggplot objects, one per graph (e.g. per time window).

Examples

```
network <- build_network(
  nodes = subset(Nodes_stagflation, source_type == "Stagflation"),
  directed_edges = Ref_stagflation,
  source_id = "source_id",
  target_id = "target_id",
  method = "similarity",
  edges_threshold = 1
)
network <- add_clusters(
  network,
  clustering_method = "leiden",
  objective_function = "modularity",
  seed = 123,
  verbose = FALSE
```

```
)
set.seed(1)
network <- layout_networks(network, node_id = "source_id", layout = "fr")
network <- color_networks(network, column_to_color = "cluster_leiden")
if (requireNamespace("ggrepel", quietly = TRUE)) {
  plot_networks(
    network,
    cluster_label_column = "cluster_leiden",
    node_size_column = NULL
  )
}
```

`prepare_label_networks` *Position the labels of clusters in network and alluvial plots*

Description

`prepare_label_networks()` computes, for each category of `cluster_label_column`, the mean coordinates of its nodes, where the label of the cluster will be drawn by `plot_networks()`: `label_x` and `label_y` are the centre of gravity of the cluster.

`prepare_label_alluvial()` chooses the time window where the label of each cluster of an alluvial will be drawn: the median window of the windows where the cluster exists. The chosen row gets the label in a `label_x` column, which `plot_alluvial()` uses.

Usage

```
prepare_label_networks(
  graphs,
  x = "x",
  y = "y",
  cluster_label_column = "cluster_label"
)

prepare_label_alluvial(
  alluv_dt,
  window_column = "window",
  cluster_label_column = "cluster_label"
)
```

Arguments

`graphs`	A `tbl_graph` or a list of `tbl_graphs` with coordinates (see `layout_networks()`).
`x, y`	Character scalars, the node columns with the coordinates (`"x"` and `"y"` by default).

`cluster_label_column`
: Character scalar, the column with the labels: `"cluster_label"` (`name_clusters()`) by default, or a cluster column.

`alluv_dt`
: A data frame created with `networks_to_alluv()`.

`window_column`
: Character scalar, the column of `alluv_dt` with the time windows (`"window"` by default, as created by `networks_to_alluv()`).

Value

`prepare_label_networks()`: the same object as `graphs` with node columns `label_x` and `label_y`. `prepare_label_alluvial()`: `alluv_dt` with a `label_x` column, holding the label on the rows of the chosen window and missing elsewhere.

Examples

```
network <- build_network(
  nodes = subset(Nodes_stagflation, source_type == "Stagflation"),
  directed_edges = Ref_stagflation,
  source_id = "source_id",
  target_id = "target_id",
  method = "similarity",
  edges_threshold = 1
)
network <- add_clusters(
  network,
  clustering_method = "leiden",
  objective_function = "modularity",
  seed = 123,
  verbose = FALSE
)
set.seed(1)
network <- layout_networks(network, node_id = "source_id", layout = "fr")
network <- prepare_label_networks(network, cluster_label_column = "cluster_leiden")
unique(tidygraph::as_tibble(network, active = "nodes")[c("cluster_leiden", "label_x", "label_y")])
```

Ref_stagflation *References cited by the documents explaining the 1970s US stagflation*

Description

The direct citations of the corpus of Goutsmedt (2021): one row per link between a document of Nodes_stagflation explaining the stagflation and a document it cites. It is the bipartite relation that `build_network()` and `build_dynamic_networks()` project into a bibliographic coupling network.

Usage

```
Ref_stagflation
```

Format

A data frame with 4,416 rows and 6 variables:

source_id Identifier of the citing document.

target_id Identifier of the cited document.

target_author Author of the cited document.

target_year Year of publication of the cited document.

target_title Title of the cited document.

target_journal Journal of the cited document, for articles.

Source

Goutsmedt (2021).

References

Goutsmedt A (2021). "From the Stagflation to the Great Inflation: Explaining the US Economy of the 1970s." *Revue d'Économie Politique*, **131**(3), 557–582. https://www.cairn.info/revue-d-economie-politique-2021-3-page-557.htm.

Index

∗ **datasets**
  Authors_stagflation, 10
  Edges_coupling, 22
  Nodes_coupling, 45
  Nodes_stagflation, 46
  Ref_stagflation, 54

add_clusters, 3
add_clusters(), *7*, *17*, *20, 21*, *30*, *33*
add_node_roles, 6
add_node_roles(), *6*, *39*
Authors_stagflation, 10, *47*

backbone::backbone_from_projection(), *12*, *17*, *20*
biblionetwork::biblio_coupling(), *17*
biblionetwork::coupling_angle(), *17*, *23*
biblionetwork::coupling_jaccard(), *17*
biblionetwork::coupling_similarity(), *17*
biblionetwork::coupling_strength(), *17*
biblionetwork::coupling_vladutz(), *17*
biblionetwork::entity_network(), *12*, *17*, *19*
build_dynamic_networks, 10
build_dynamic_networks(), *3*, *8, 9*, *19, 20*, *23*, *26, 27*, *33*, *54*
build_network, 18
build_network(), *17*, *27*, *39*, *52*, *54*

color_alluvial *(*color_networks*)*, 21
color_alluvial(), *45*, *47, 48*
color_networks, 21
color_networks(), *5*, *44*, *51, 52*

Edges_coupling, 22, *46*
extract_tfidf, 23
extract_tfidf(), *39, 40*, *42*

filter_components, 26
filter_components(), *13*, *20*, *28*

ggraph::create_layout(), *27, 28*

igraph::cluster_fast_greedy(), *5*
igraph::cluster_infomap(), *4, 5*
igraph::cluster_leiden(), *4, 5*
igraph::cluster_louvain(), *5*
igraph::cluster_walktrap(), *5*

layout_networks, 27
layout_networks(), *51*, *53*

merge_diagnostics, 30
merge_diagnostics(), *34*, *50*
merge_dynamic_clusters, 32
merge_dynamic_clusters(), *6, 7*, *17*, *21*, *30, 31*, *38*, *44*, *48*
minimize_crossing_alluvial, 35
minimize_crossing_alluvial(), *45*, *47, 48*

name_clusters, 38
name_clusters(), *6*, *34*, *44*, *51, 52*, *54*
networks_to_alluv, 43
networks_to_alluv(), *21, 22*, *34*, *36*, *47, 48*, *54*
Nodes_coupling, *23*, 45
Nodes_stagflation, *10*, 46, *46*, *54*

plot_alluvial, 47
plot_alluvial(), *34*, *37*, *45*, *53*
plot_merge_diagnostics, 49
plot_merge_diagnostics(), *30, 31*
plot_networks, 51
plot_networks(), *53*
prepare_label_alluvial *(*prepare_label_networks*)*, 53
prepare_label_alluvial(), *45*, *47, 48*
prepare_label_networks, 53
prepare_label_networks(), *51, 52*

Ref_stagflation, *23*, *46*, 54

set.seed(), *28*
stats::hclust(), *36*

tidygraph::centrality, *39*
tidygraph::tbl_graph(), *46*
tidytext::get_stopwords(), *24*
tidytext::unnest_tokens(), *24*

uwot::umap(), *29*

withr::local_seed(), *4*